МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
КАФЕДРА ПРИКЛАДНОЇ МАТЕМАТИКИ ТА ІНФОРМАТИКИ

СХЕМОКОНСПЕКТ ЛЕКЦІЙ

з навчальної дисципліни

«СУЧАСНІ ТЕХНОЛОГІЇ ПРОГРАМУВАННЯ»

для студентів ОС «бакалавр» спеціальності
122 Комп'ютерні науки

Луцьк – 2024

УДК 004.42(076)
С 92

Схемоконспект лекцій з навчальної дисципліни «Сучасні технології програмування» для студентів ОС «бакалавр» спеціальності 122 Комп'ютерні науки / уклад. Є.О. Башков, А.О. Нікітенко. – Луцьк: ДонНТУ, 2024. – 423 с.

Навчальна дисципліна “Сучасні технології програмування» є невід'ємною складовою сучасної підготовки студентів спеціальності 122 «Комп'ютерні науки». У контексті стрімкого розвитку сфери комп'ютерних наук обізнаність майбутнього фахівця у мовах та технологіях побудови систем штучного інтелекту ІТ, що базуються на мові Python та відповідних бібліотек є ключовим фактором подальшого успіху.

Вивчення дисципліни «Сучасні технології програмування" допомагає студентам засвоїти основні принципи розробки прикладного програмного забезпечення, основи технології структурного підходу до програмування, концепцію і складові частини об'єктно-орієнтованого програмування, володіти сучасними технології програмування на об’єктно-орієнтованих високого рівня мовах програмування (Python), мати уявлення щодо самостійної розробкою програмного забезпечення для подальшого виконання курсових і випускних кваліфікаційних робіт.

Укладачі: Є.О. Башков, професор, д.т.н., професор кафедри ПМІ
А.О. Нікітенко, асистент кафедри ПМІ

Рецензент: С.О. Ковальов, доцент, к. т. н., доцент кафедри електронної техніки

Відповідальний за випуск: Н.О. Маслова, доцент, к. т. н., зав. каф. ПМІ
Затверджено навчально-методичним відділом ДонНТУ, протокол № 5 від 16.01.2024 р.

Розглянуто на засіданні кафедри ПМІ, протокол № 13 від 27.12.2023 р.

ЗМІСТ

ЗМІСТ

ЛЕКЦІЯ 01

1. Програма дисципліни.
2. Історія ОТ.
3. Історія програмування.
4. Сучасні мови програмування. Класифікація.
5. Мова Python. Загальні відомості

ОБСЯГ УЧБОВОЇ РОБОТИ

Всього **6** кредитів, **180** годин

- **Аудиторних**
  - Лекцій **32** годин / **16** лекцій
  - Практичних **32** годин / **16** занять
- **Розрахункова робота (РР)**
- **Іспит**

ПРОГРАМА. Тематика лекцій

Тема 1. Високорівнева мова програмування Python

Тема 2. Базові пакети Python

Тема 3. Технології програмування

ПРОГРАМА.

Тематика практичних занять

1. Мова програмування Python.
2. Базові пакети

Тематика розрахункової роботи

Створення програмного пакету для вирішення

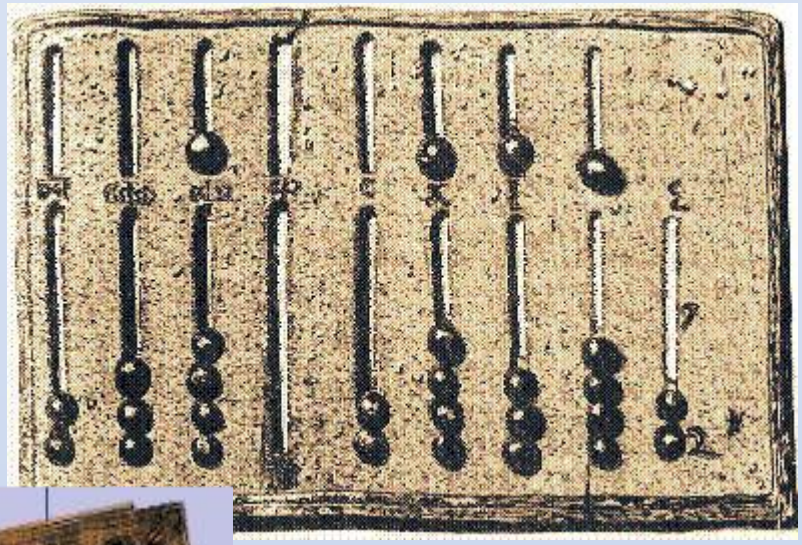
1. Систем лінійних рівнянь.
2. Систем нелінійних рівнянь.
3. Систем диференціальних рівнянь.

МЕТА КУРСУ

отримання знань та навиків, які необхідні для проектування та розробки програмних додатків на основі сучасних технологій програмування.

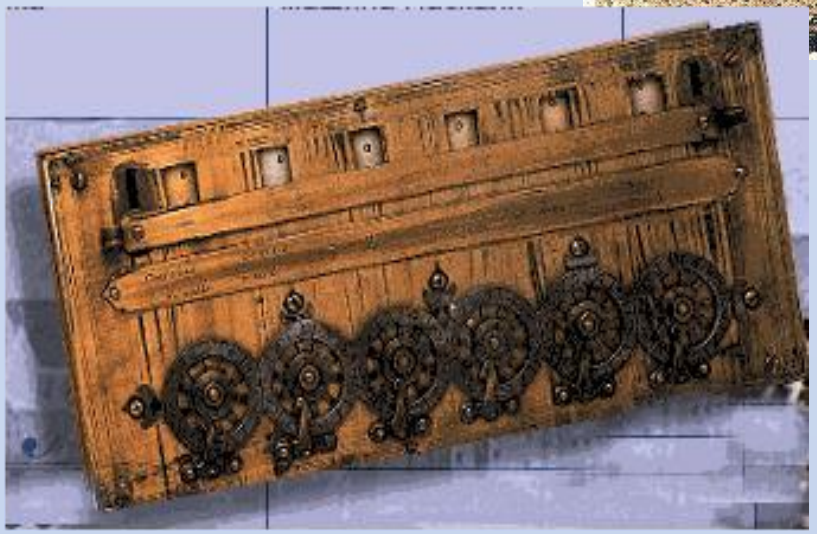
ВВЕДЕННЯ. Історія ОТ

Комп'ютер (обчислювальна машина, ЕОМ, ЦОМ) — *програмно* - керований пристрій для обробки інформації.



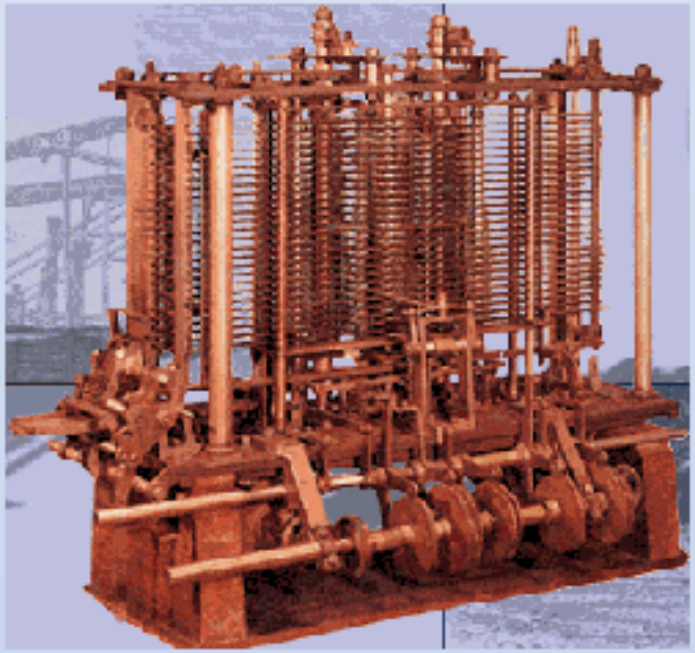
Абак (?)

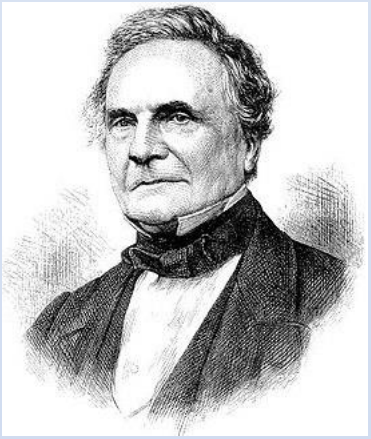
Блез Паскаль (1623- 1662)



ВВЕДЕННЯ. Історія ОТ

Чарльз Беббідж (1792-1881)

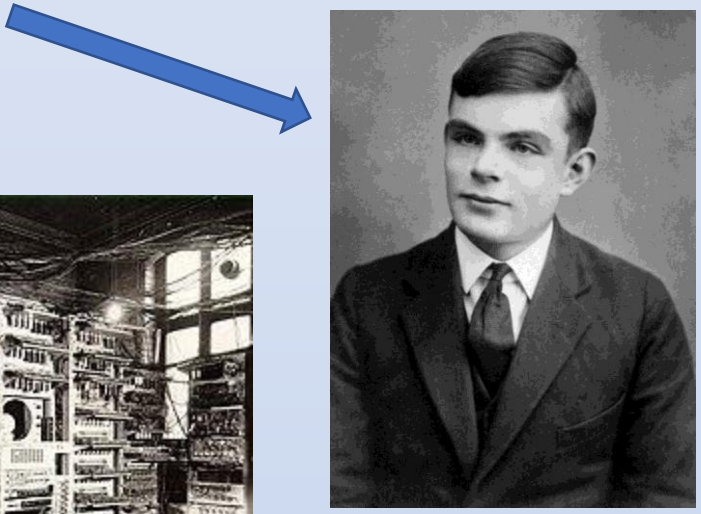


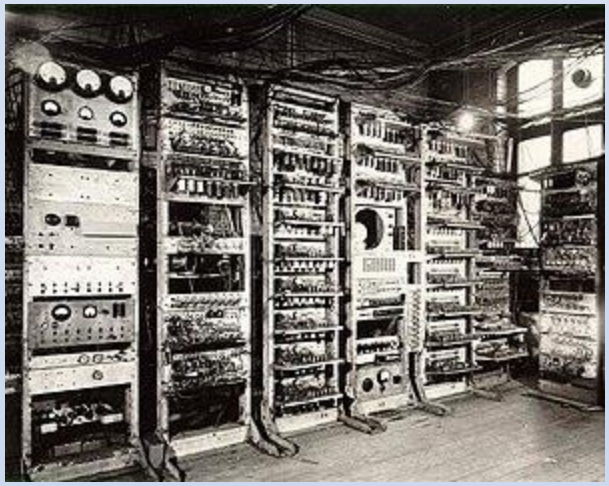


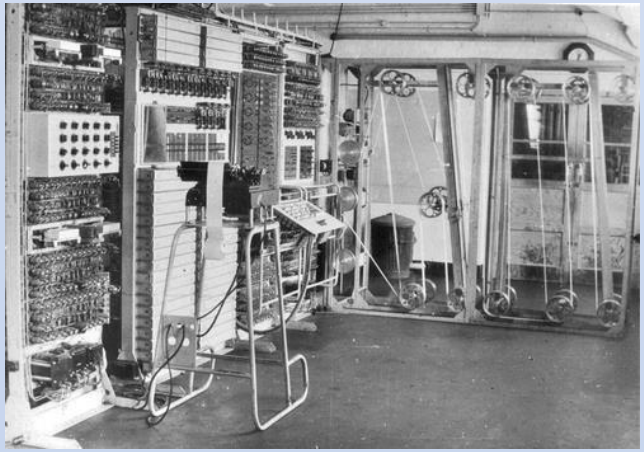
ВВЕДЕННЯ. Історія ОТ

Алан Тьюрінг (1912- 1954)

Colossus, Mark



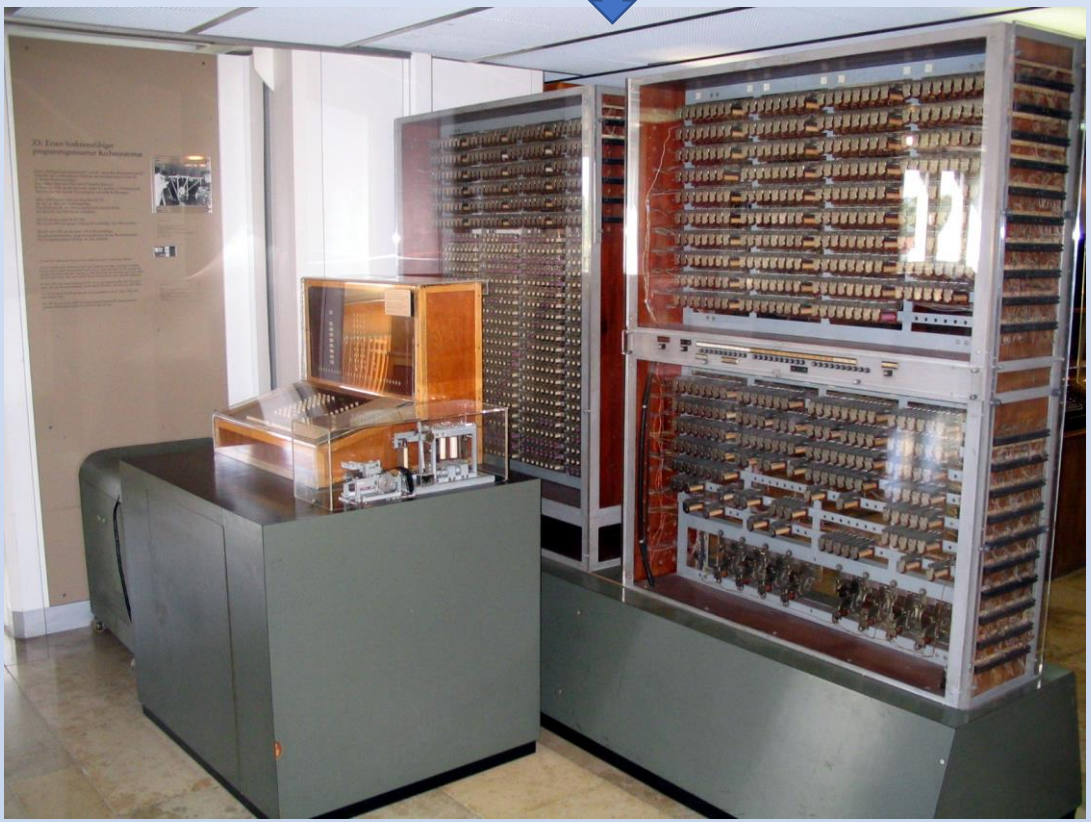




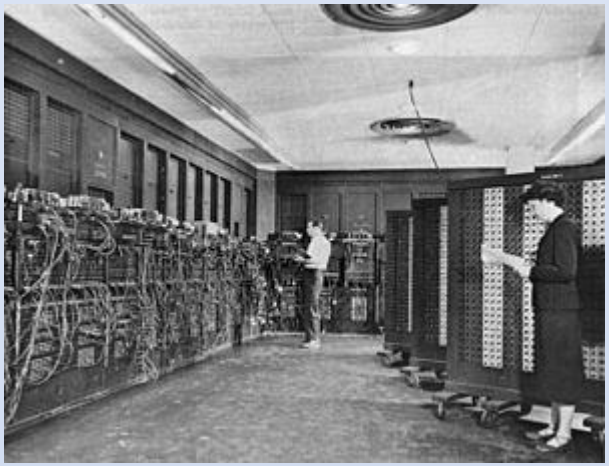
ВВЕДЕННЯ. Історія ОТ

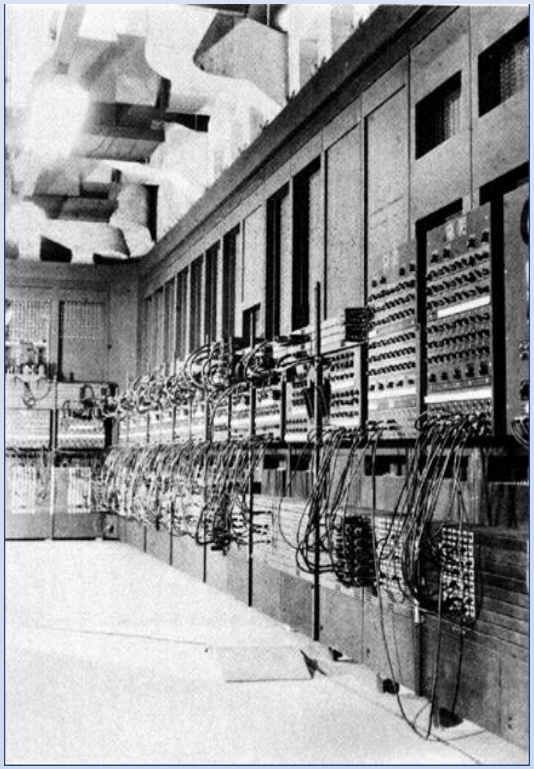
Конрад Цузе (1910- 1995)

Z1 (1938), Z2, Z3



ВВЕДЕННЯ. Історія ОТ





Джон фон Нейман (1903-1957

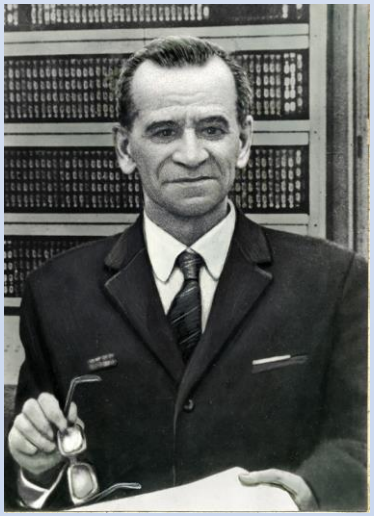
Говард Ейкєн (Harvard Mark I, II, III)

Джон Еккерт, Джон Моклі (ENIAC)

ВВЕДЕННЯ. Історія ОТ

Сергій Лебедєв (1902-1974)

МЭСМ, БЭСМ



Історія ОТ дивись https://uk.wikipedia.org/wiki/компютер

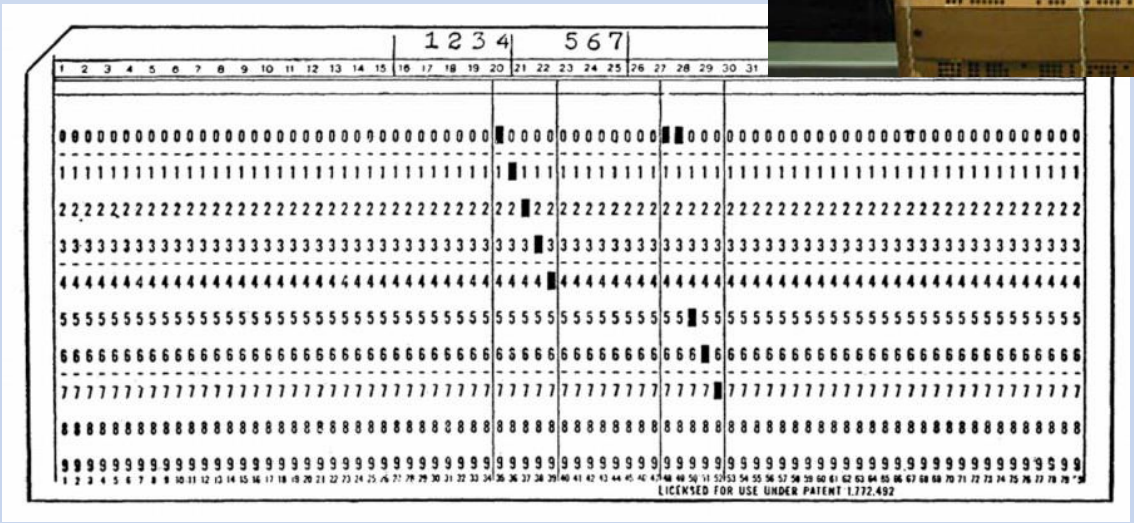
http://ua.uacomputing.com/stories/mesm/

ВВЕДЕННЯ. Історія програмування

Мова програмування — штучна формальна система, засобами якої можна виражати алгоритми. Мова програмування визначає набір лексичних, синтаксичних, та семантичних правил, що задають зовнішній вигляд програми і дії, які виконує виконавець (комп'ютер) під її управлінням

ВВЕДЕННЯ. Історія програмування

Жозеф Жаккард (1752-1843)



ВВЕДЕННЯ. Історія програмування

Ада Лавлейс (1815-1852)

Sketch of

The Analytical Engine

Invented by Charles Babbage

By L. F. MENABREA
of Turin, Officer of the Military Engineers

from the *Bibliothèque Universelle de Genève*, October, 1842, No. 82

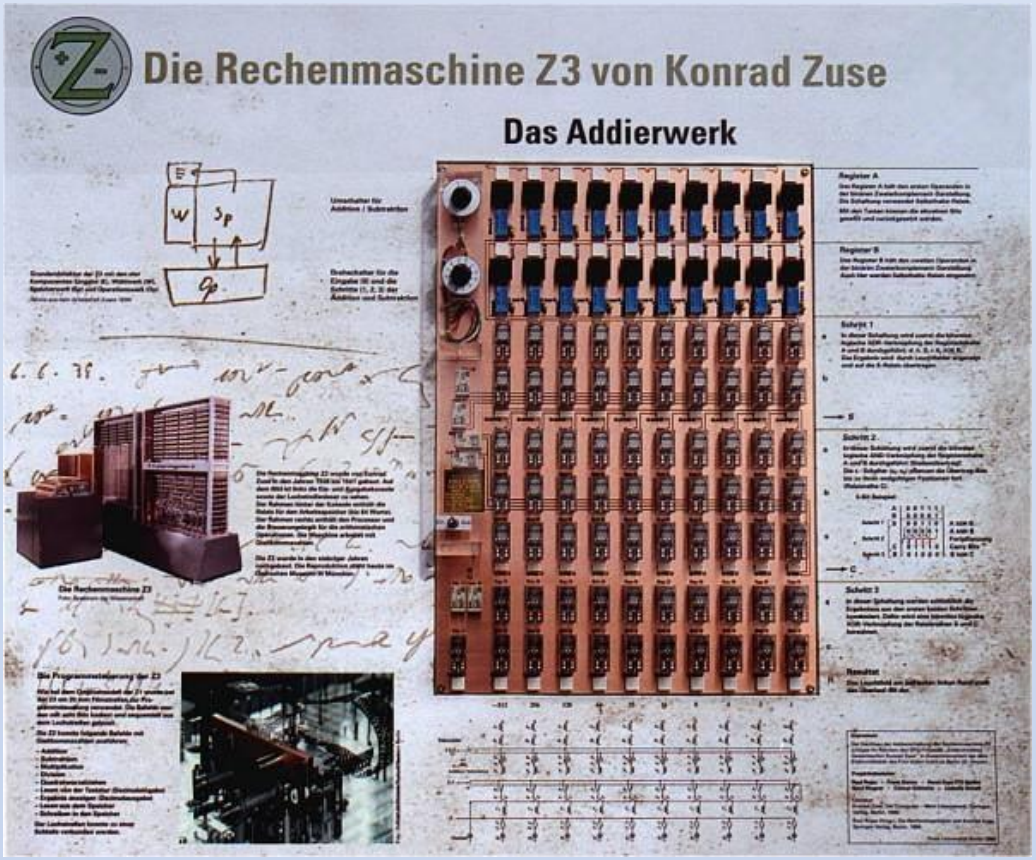
With notes upon the Memoir by the Translator
ADA AUGUSTA, COUNTESS OF LOVELACE

Columns on which are inscribed the primitive data	Number of the operations	Cards of the operations: No. of the Operation-cards	Cards of the operations: Nature of each operation	Variable cards: Columns acted on by each operation	Variable cards: Columns that receive the result of each operation	Variable cards: Indication of change of value on any column	Statement of results
${}^1V_0 = m$	1	1	×	${}^1V_0 \times {}^1V_4 =$	${}^1V_6$	$\left\{ {}^1V_0 = {}^1V_0 \right.$ $\left. {}^1V_4 = {}^1V_4 \right\}$	${}^1V_6 = mn'$
${}^1V_1 = n$	2	"	×	${}^1V_3 \times {}^1V_1 =$	${}^1V_7$	$\left\{ {}^1V_3 = {}^1V_3 \right.$ $\left. {}^1V_1 = {}^1V_1 \right\}$	${}^1V_7 = m'n$
${}^1V_2 = d$	3	"	×	${}^1V_2 \times {}^1V_4 =$	${}^1V_8$	$\left\{ {}^1V_2 = {}^1V_2 \right.$ $\left. {}^1V_4 = {}^0V_4 \right\}$	${}^1V_8 = dn'$
${}^1V_3 = m'$	4	"	×	${}^1V_5 \times {}^1V_1 =$	${}^1V_9$	$\left\{ {}^1V_5 = {}^1V_5 \right.$ $\left. {}^1V_1 = {}^0V_1 \right\}$	${}^1V_9 = d'n$
${}^1V_4 = n'$	5	"	×	${}^1V_0 \times {}^1V_5 =$	${}^1V_{10}$	$\left\{ {}^1V_0 = {}^0V_0 \right.$ $\left. {}^1V_5 = {}^0V_5 \right\}$	${}^1V_{10} = d'm$
${}^1V_5 = d'$	6	"	×	${}^1V_2 \times {}^1V_3 =$	${}^1V_{11}$	$\left\{ {}^1V_2 = {}^0V_2 \right.$ $\left. {}^1V_3 = {}^0V_3 \right\}$	${}^1V_{11} = dm'$
	7	2	−	${}^1V_6 - {}^1V_7 =$	${}^1V_{12}$	$\left\{ {}^1V_6 = {}^0V_6 \right.$ $\left. {}^1V_7 = {}^0V_7 \right\}$	${}^1V_{12} = mn' - m'n$
	8	"	−	${}^1V_8 - {}^1V_9 =$	${}^1V_{13}$	$\left\{ {}^1V_8 = {}^0V_8 \right.$ $\left. {}^1V_9 = {}^0V_9 \right\}$	${}^1V_{13} = dn' - d'n$
	9	"	−	${}^1V_{10} - {}^1V_{11} =$	${}^1V_{14}$	$\left\{ {}^1V_{10} = {}^0V_{10} \right.$ $\left. {}^1V_{11} = {}^0V_{11} \right\}$	${}^1V_{14} = d'm - dm'$
	10	3	÷	${}^1V_{13} \div {}^1V_{12} =$	${}^1V_{15}$	$\left\{ {}^1V_{13} = {}^0V_{13} \right.$ $\left. {}^1V_{12} = {}^1V_{12} \right\}$	${}^1V_{15} = \frac{dn' - d'n}{mn' - m'n} = x$
	11	"	÷	${}^1V_{14} \div {}^1V_{12} =$	${}^1V_{16}$	$\left\{ {}^1V_{14} = {}^0V_{14} \right.$ $\left. {}^1V_{12} = {}^0V_{12} \right\}$	${}^1V_{16} = \frac{d'm - dm'}{mn' - m'n} = y$
1	2	3	4	5	6	7	8

ВВЕДЕННЯ. Історія програмування

Конрад Цузе (1945)

мова **PlanKalKul - обчислювателль планів**



ВВЕДЕННЯ. Історія програмування

Тоні Брукнер (1951). Manchester Mark I - Мова Autocode

An Example

Tabulate Sievert's integral $\int_0^y e^{-a \sec x} dx$ for $y = 1(1)90^{o}$ and particular values of a. The method adopted is to tabulate the integrand for $x = 0(\frac{1}{2})\ 90^{o}$ and calculate the integral step-by-step using Simpson's rule to evaluate the increments;

$$\int_0^{y+h} = \int_0^y + \frac{h}{6}\left[f(y) + 4f\left(y + \frac{h}{2}\right) + f(y+h)\right]$$

where $h = \frac{\pi}{360}$

```
 8                f → 180
 2                h = π/360                    (5 m
10                s = 0(1)179                     1
12                b = φ cos (sh)               (23 m
 4              f_s = φ exp (-a/b)             (23 m
 4                repeat                          4
 4            f_180 = 0                           4
 2                y = 0                           4
 4                r = 1(1)90                      1
 7                newline                      (90 m
 2                print (r)2,0                (150 m
11                space                        (60 m
10                s = 2r - 1                      19
 9                f = f_(s-1) + 4f_s + f_(s+1)    22
 7                y = y + hf/3                 (5 m
 4                print (y) 1,6               (333 m
                  repeat                          4
                  end
```

ВВЕДЕННЯ. Історія програмування

Джон Бекус (1954).

Мова FORTRAN (!!! Fortran 2018)

…..

1958 LISP

1958 ALGOL

1959 COBOL

1964 BASIC

1971 PASCAL

1972 C ……

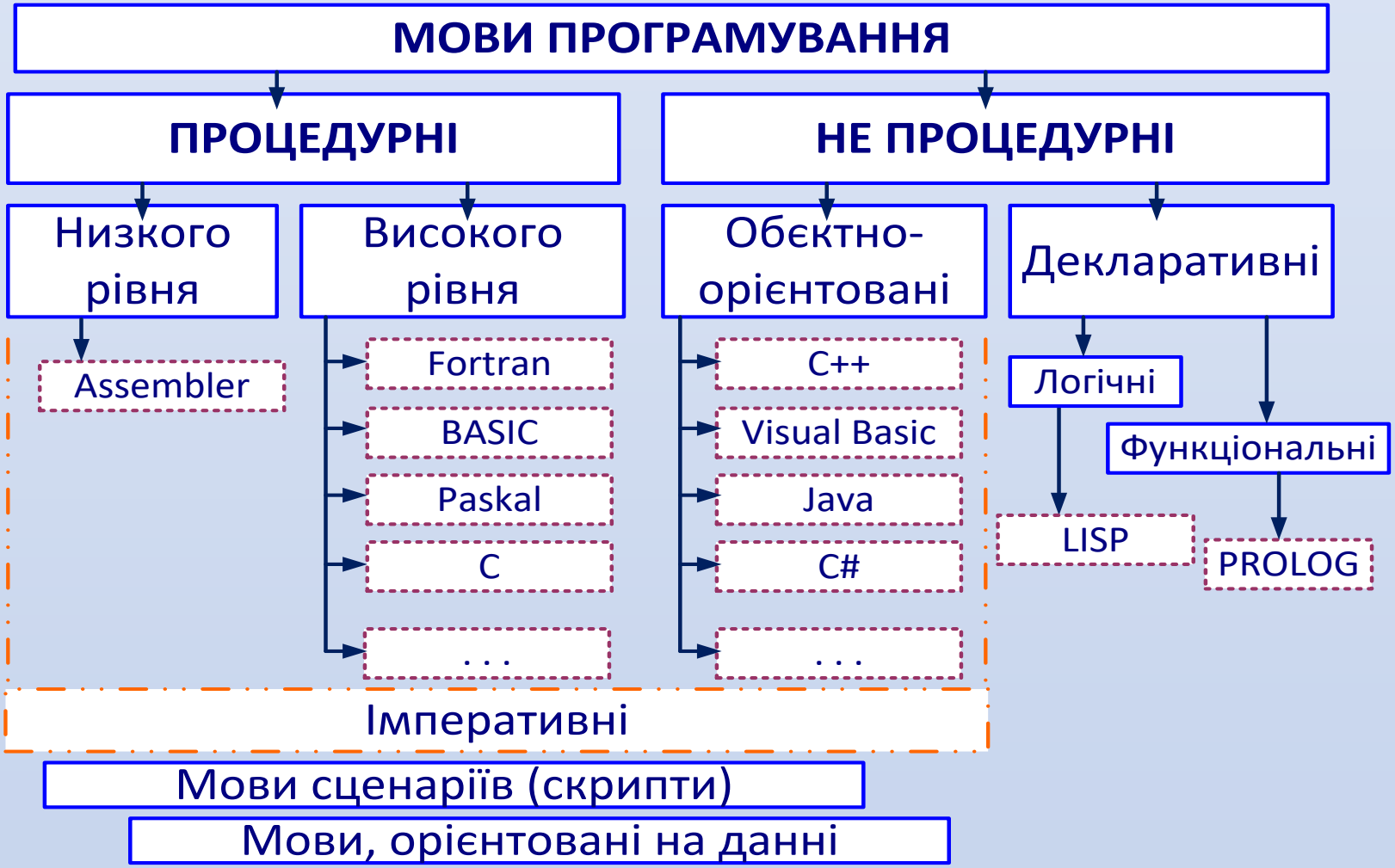
Історія програмування дивись:

https://uk.wikipedia.org/wiki/мова_програмування

https://ru.wikipedia.org/wiki/хронология_языков_программирования

ВВЕДЕННЯ. Мови програмування

Класифікація (спрощена !!!)



! Більш ніж 2000 мов програмування

ВВЕДЕННЯ. Мови програмування

Процедурні мови - мови високого рівня, в яких використовується метод розбиття програм на окремі пов'язані між собою модулі - підпрограми (процедури і функції). Компоненти мови складаються з послідовності операторів, які використовують бібліотечні процедури і функції.

Непроцедурні мови – мови високого рівня орієнтовані на використання «технологічне» програмування – маніпулювання деяким сутностями предметної області.

ВВЕДЕННЯ. Мови програмування

Мови низького рівня – орієнтовані на певний тип процесора, враховують його архітектурні особливості (програмування в кодах, **assembler, macroassembler**).

Мови високого рівня – не враховують особливості конкретного процесора, програми на мовах високого рівня достатньо легко переносяться с одної архітектури на іншу.

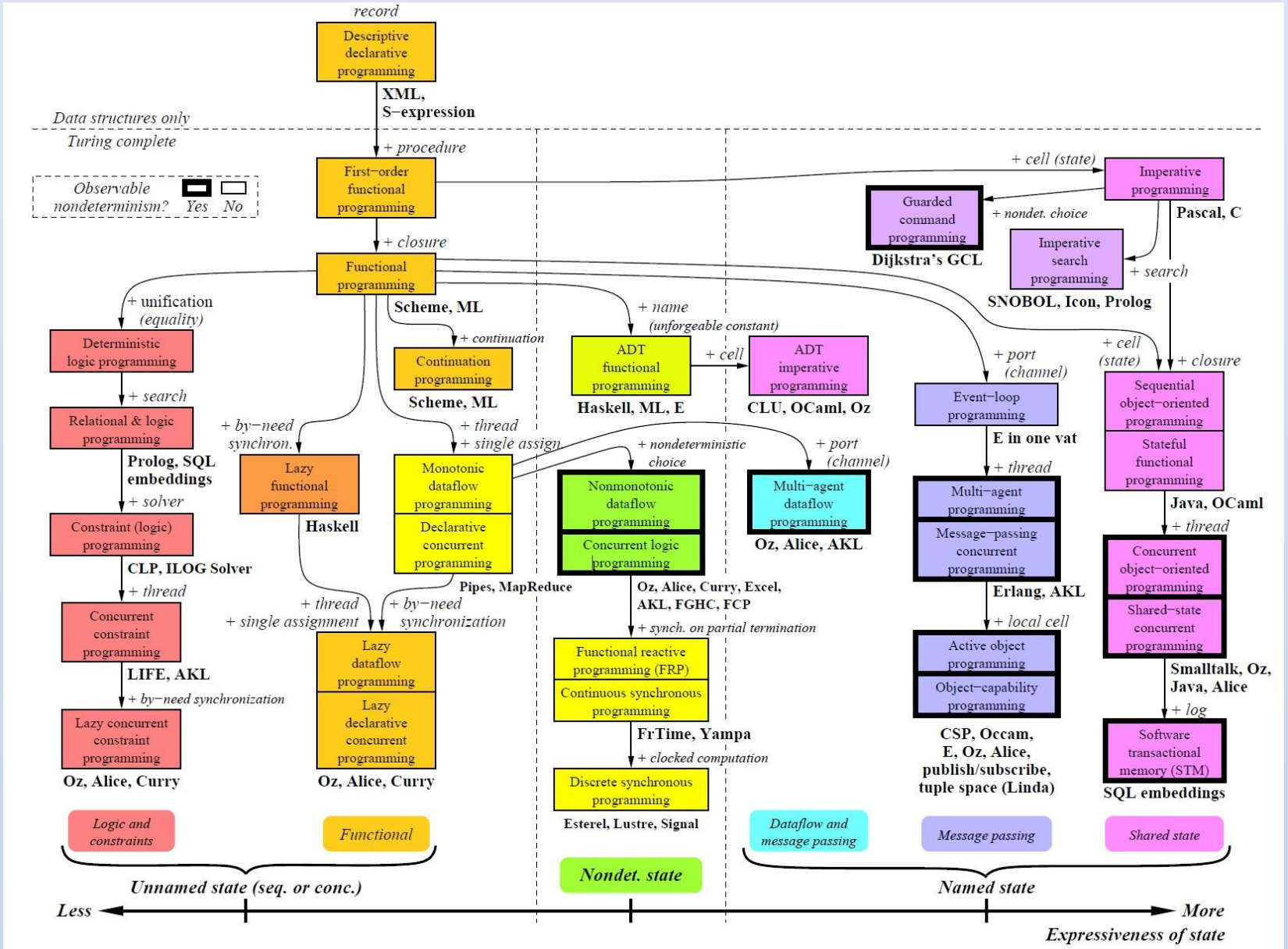
ВВЕДЕННЯ. Мови програмування

Об'єктно - орієнтовні мови – використовують множину програмних об'єктів – сутностей, що об'єднують в собі дані (поля) та дії (методи) (C++, Python, …).

Декларативні мови – описують результат, який потрібно отримати, замість послідовності операцій з отримання цього результату (SQL, HTML).

Імперативні мови – детально описують деякий алгоритм отримання результатів.

ВВЕДЕННЯ. Мови програмування



Peter Van Roy. Programming Paradigms for Dummies: What Every Programmer Should Know

ВВЕДЕННЯ. Мови програмування

Реально використовується ≅ 100 мов

	C	C++	C#	Java	Python	Delphi	Ruby	PHP	Small talk	Lisp
Імперативні	+	+	+	+	+	+	+	+	+	+
Декларативні	-	-	-	-	+	-	+	-	+	+
Функціональні	-	-	-	-	+	-	+	-	+	+
Об`єктні	-	+	+	+	+	+	+	+	+	+
Логічні	-	-	-	-	-	-	-	-	+	+
Розподілені	-	-	-	-	-	-	-	-	+	+

Дивись:

https://www.youtube.com/watch?v=Og847HVwRSI&feature=youtu.be

ВВЕДЕННЯ. Python

Python (Пайтон, 1990) **-** інтерпретована, об'єктно - орієнтовна мова програмування високого рівня .

Python підтримує модулі та пакети. Інтерпретатор **Python** та стандартні бібліотеки доступні на всіх основних платформах.

Python підтримується : об'єктно – орієнтоване, процедурне, функціональне та аспектно-орієнтовне програмування.

Офіційний сайт : https://www.python.org/

ВВЕДЕННЯ. Python

Пайтон Дзен (*скорченно https://peps.python.org/pep-0008/*)

- Гарне краще за потворне. Явне краще за неявне.
- Просте краще за складне. Складне краще за заплутане.
- Легкість читання має значення. Особливі випадки не є настільки особливими, щоб порушувати правила.
- Помилки ніколи не повинні проходити непомітно. Якщо їх приховування не прописано явно.
- Має бути один — і, бажано, ***тільки*** один — очевидний спосіб зробити це.
- Зараз — краще, ніж ніколи. Хоча ніколи, найчастіше, — краще, ніж ***просто*** зараз.
- Якщо реалізацію важко пояснити — задум поганий. Якщо реалізацію легко пояснити — *можливо*, задум добрий.

ВВЕДЕННЯ. Python

Розвиток:

Python 1.0 (1980)

Python 2.0 (2000)

Python 3.0 (2008) // зворотне несумісний

Інструментарій:

Anaconda (MIT) – open source дистрибутив

- **Spyder**
- **JupyterLab**
- **Jupyter Notbook**

https://anaconda.org/

ВВЕДЕННЯ. Python

Переваги:

- Інтерпретована мова програмування.
- Динамічна типізація.
- Модульність.
- Вбудована підтримка Unicode.
- Об'єктно - орієнтоване програмування.
- Автоматична збірка мусору.
- Інтеграція з C / C++.
- Кросплатформеність.

Недоліки:

- (!) Інтерпретована мова програмування.

Контрольні питання

- Надайте визначення мови програмування
- Надайте визначення імперативної мови програмування. Наведіть приклади.
- Надайте визначення декларативної мови програмування. Наведіть приклади.
- Надайте визначення мови програмування низького рівня. Наведіть приклади.
- Надайте визначення мови програмування високого рівня. Наведіть приклади.
- Надайте визначення об'єктно - орієнтованої мови програмування Наведіть приклади.
- Надайте базові властивості мови програмування Python.

The END
Лекція 01

ЛЕКЦІЯ 02. PYTHON #1

1. Ідентифікатори
2. Запис програм
3. Змінні, типи, типізація
4. Типи даних: числа
5. Оператори, операції
6. Логічний тип даних, логічні вирази
7. Алгоритмічні структури: слідування, розгалуження, цикл

ІДЕНТИФІКАТОРИ

Ідентифікатор **– деяке ім'я, яке використовується для ідентифікації об'єкту:** ***змінної, функції, класу, модуля …***

Ідентифікатор може містити тільки символи:

- Літери в нижньому регістрі ***a … z***
- Літери в верхньому регістрі ***A … Z***
- Цифри ***0 … 9***
- Нижнє підкреслення _

!! Не може починатися з цифри

!! Не може співпадати з зарезервованими словами

ЗАПИС ПРОГРАМИ

!!! Блоки коду відокремлюються за допомогою рядкового відступу

```
if s>0 :
    print (“YES”)
else :
    print (“NO”)
```

!!! Жорстка вимога

Коментар **–** фрагмент тексту програми, що ігнорується при виконанні.

Коментар в рядку починаються з символу **#**

```
I = 0   #index initialization
```

ЗМІННІ, ТИПИ, ТИПІЗАЦІЯ

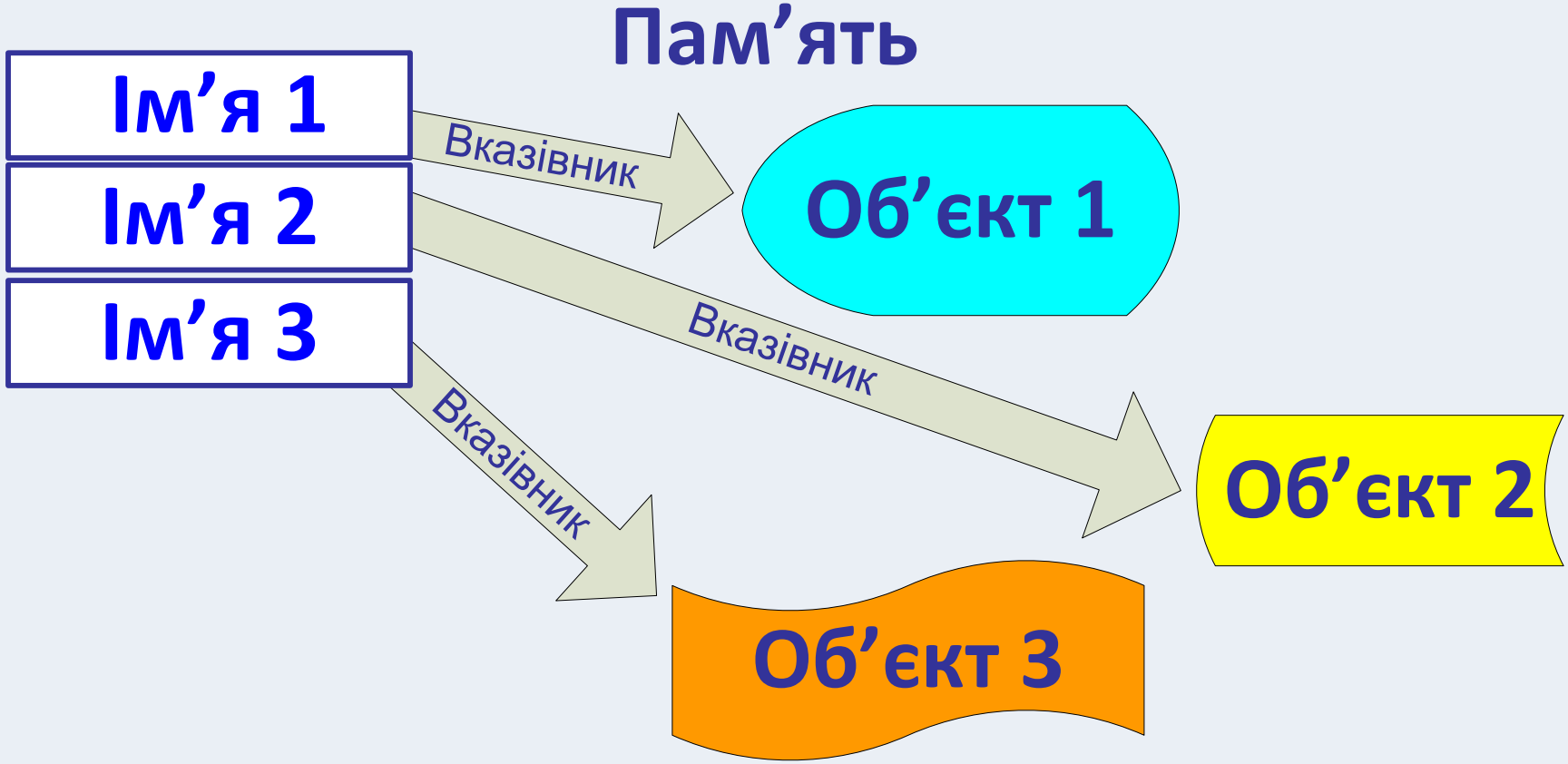
Змінна - це ім'я, яке посилається на деякий об'єкт (значення) в пам'яті комп'ютера.

A=3

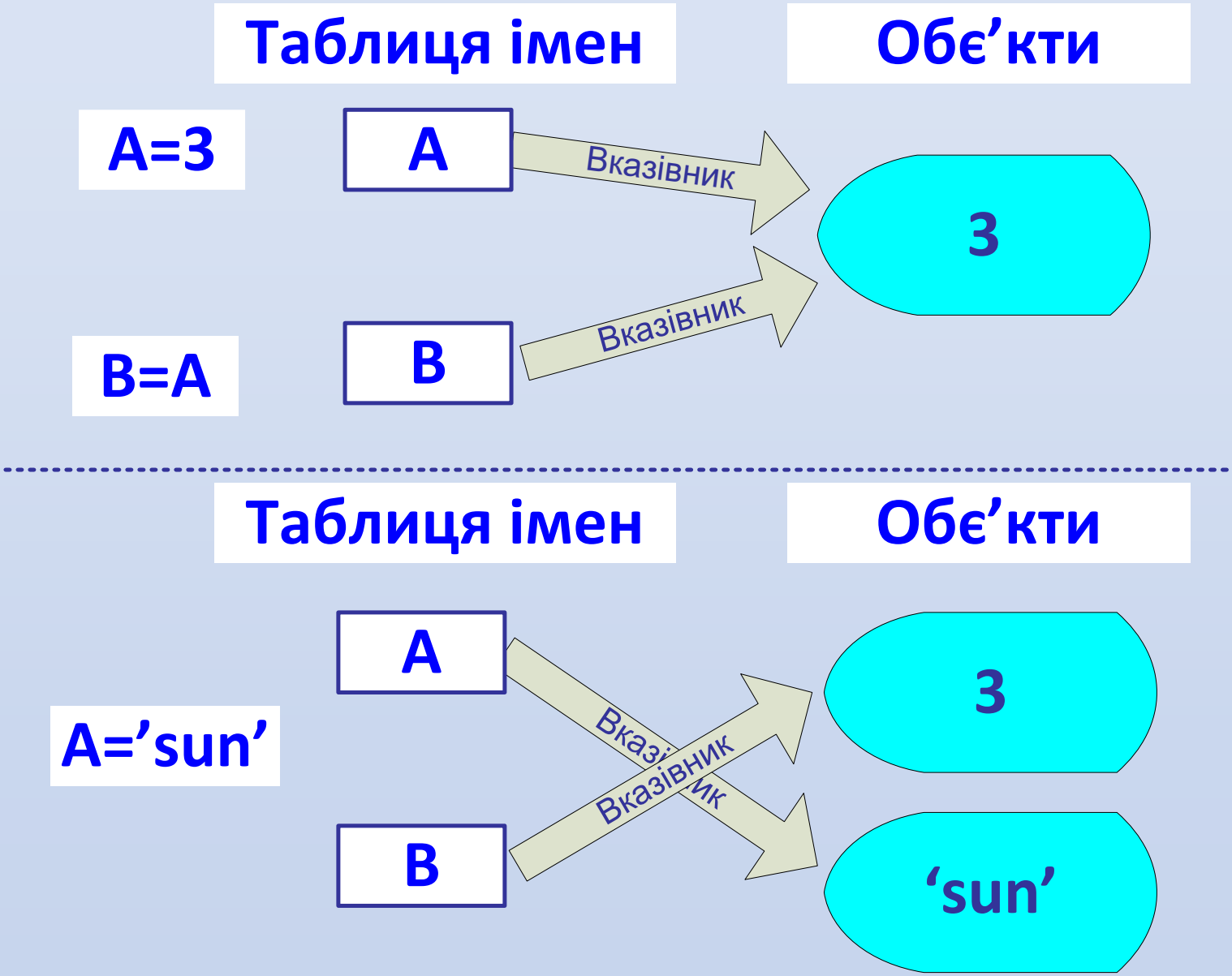
Змінні **створюються** при виконанні операції присвоювання значення.
Для використання в виразах змінна повинна мати значення (**ініціалізована !**).
Під час обчислення значень деякого виразу ім'я змінної **заміщується** її значенням.

ЗМІННІ, ТИПИ, ТИПІЗАЦІЯ

!!! Змінна = Ім’я



ЗМІННІ, ТИПИ, ТИПІЗАЦІЯ



ЗМІННІ, ТИПИ, ТИПІЗАЦІЯ

Тип об'єкту – множина значень та множина операцій на цих значеннях. Тип визначає можливі значення та їх сенс, способи зберігання цих значень, можливі операції над значеннями.

Типізація – операція визначення типу інформаційній сутності (об'єкту)

	СТАТИЧНА	ДИНАМІЧНА
СИЛЬНА	C#	Python
СЛАБКА	C	JavaScript

ЗМІННІ, ТИПИ, ТИПІЗАЦІЯ

Статична – змінна не може змінити тип. Визначається на етапі компіляції.

Динамічна – змінна може змінити тип. Визначається при призначенні їй значення.

Сильна – не допускає виконання операції при несумісності типів.

Слабка – дозволяє виконання операції при несумісності типів. Результат ????

	СТАТИЧНА	ДИНАМІЧНА
СИЛЬНА	C#	Python
СЛАБКА	C	JavaScript

ТИП: ЧИСЛО

ОБЄ’КТ --> ЧИСЛО

ЦІЛІ

- *int*
- *Long int*
- *bool*

ДІЙСНІ - *float*

КОМПЛЕКСНІ - *complex*

None

	ЛИТЕРАЛ
Int	123 ~~-24~~ 0
~~Long~~ int	9999999999999999L
Bool	True False
Float	1.23 -123.45 -32.5E-21
Complex	3+4j 3.0 +3.0j 4.0j

python 3.0 int = long int (все long int)

ЧИСЛА. ОПЕРАТОРИ / ОПЕРАЦІЇ

Оператор	Опис
+, -	Унарні +, -. Додавання, віднімання
* , /	Множення, Ділення
// , %	Цілочисельне ділення, Залишок
**	Піднесення до степеню
<, <=, ==, >=, >, !=	Зрівняння
| , & , ^	Логічні OR, AND, XOR
<< , >>	Зсув

Старшинство операцій = старші виконуються поперед молодших

Тип результату при змішаних операціях - ранжування типів

int → float → complex

ЛОГІЧНІ ВИРАЗИ

Логічна змінна – підклас цілочисельного типу ***int,*** яка приймає значення **0** або **1** і представлена під час виводу як ***False*** та ***True***

Оператор	Опис
<, <=, ==, >=, >, !=	**Зрівняння → результат *boolean***
| , & , ^	**Логічні OR, AND, XOR**

В більш широкому сенсі:
Кожний об'єкт може бути ***True*** або ***False.***
Об'єкт ***True*** якщо він не порожній (спрощено не **0**)
Об'єкт ***False*** якщо він порожній (**None**, аналог **NULL**)

СЛІДУВАННЯ

СЛІДУВАННЯ – команди (інструкції) виконуються послідовно одна за іншою.

!!! Кінець рядка - кінець інструкції ;

!!! Відсутні дужки блоків ~~*begin end { }*~~

!!! Відступи

Допускається декілька інструкцій в один рядок

A = 2; B =3.25; c='kajfhad'

Допускається одна інструкція в декілька рядків // необхідно взяти в дужки ( ) або [ ]

РОЗГАЛУЖЕННЯ IF

РОЗГАЛУЖЕННЯ – перевірка умови (умов) і виконання відповідного блоку інструкцій

```
if <test 1> :      # умова 1
      <statements 1> # блок інструкцій 1
elif <test 2> :   # умова 2
      <statements 2> # блок інструкцій 2
else :
      <statements 3>  # блок інструкцій 3
```

ЦИКЛ WHILE

ЦИКЛ – структура, що виконує блок інструкцій доки діє деяка умова.

```
while  <test > :          # умова
      <statements 1> # блок інструкцій 1
```

Можливо додатково

```
else :                          # необов'язково
      <statements 2> # блок інструкцій 2
```

Додаткові інструкції (тільки в блоці 1)

```
break           # вихід з циклу
continue# перехід до початку циклу
pass            # пуста інструкція
```

ЦИКЛ FOR

```
for <target > in <object > :   # змінна & умова
    <statements> # блок інструкцій
```

Можливо додатково

```
else :                          # необов'язково
    <statements 2> # блок інструкцій 2
```

Додаткові інструкції (тільки в блоці)

```
break       # вихід з циклу
continue    # перехід до початку циклу
pass        # пуста інструкція
```

Контрольні запитання

- Надайте визначення змінної в мові Python. Поясніть, як визначається тип змінної в мові Python.
- Опишіть властивості змінних типу ***int*** та визначте операції з ними.
- Опишіть властивості змінних типу ***float*** та визначте операції з ними.
- Опишіть властивості змінних типу ***complex*** та визначте операції з ними.
- Опишіть властивості змінних типу ***bool*** та визначте операції з ними.
- Надайте визначення структури ***if … elif … else*** та наведіть приклади використання
- Надайте визначення структури ***while …*** та наведіть приклади використання
- Надайте визначення структури ***for … in …*** та наведіть приклади використання

The END
Лекція 02

ЛЕКЦІЯ 03. PYTHON #2

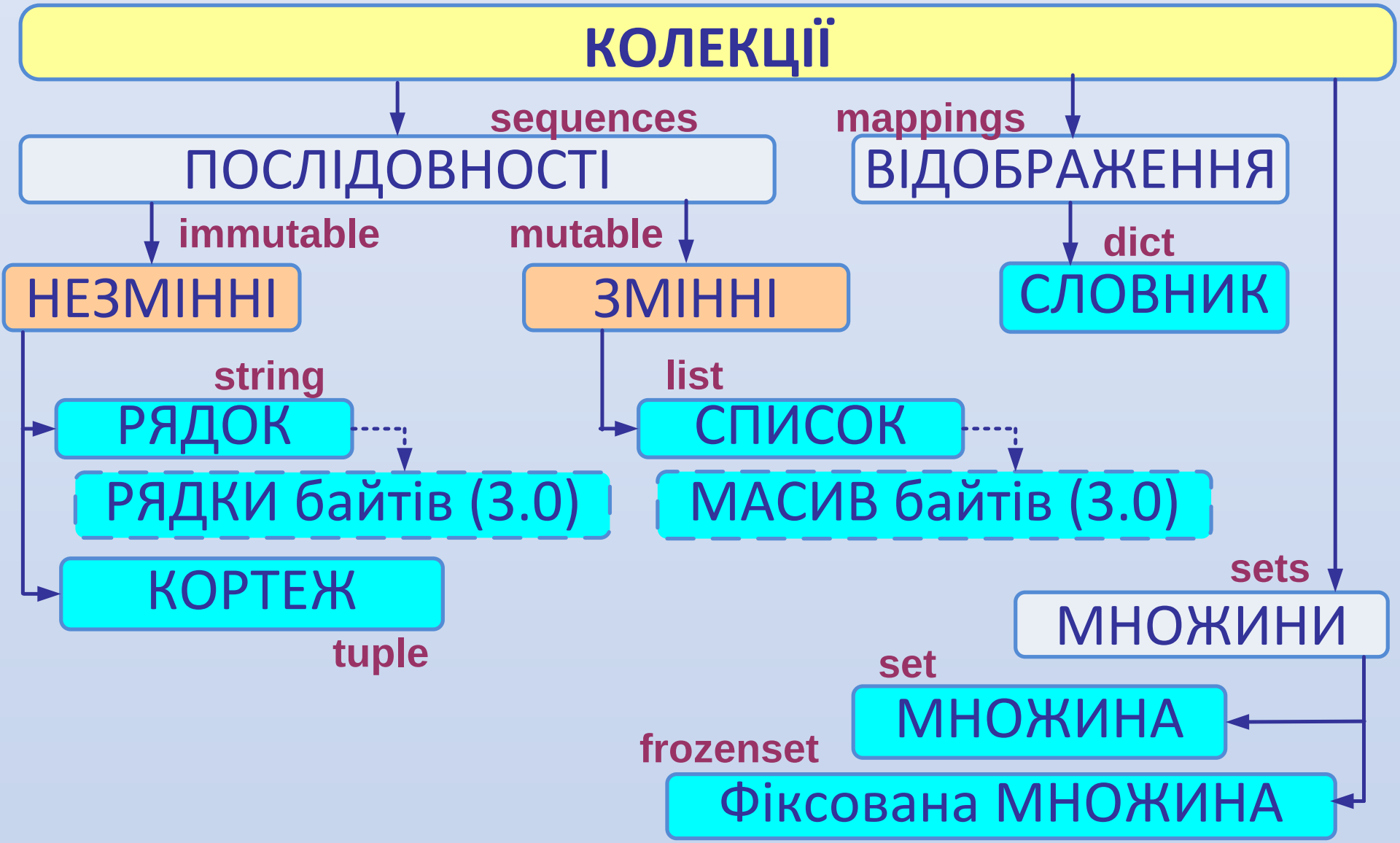
1. **Колекції: загальні відомості**
2. **Рядки (string)**
3. **Списки (list)**
4. **Словники (dict)**

КОЛЕКЦІЇ

КОЛЕКЦІЯ → **програмний об'єкт (змінна–контейнер), що зберігає значення одного або різних типів та дозволяє звертатися до цих значень а також використовувати вбудовані функції і методи.**

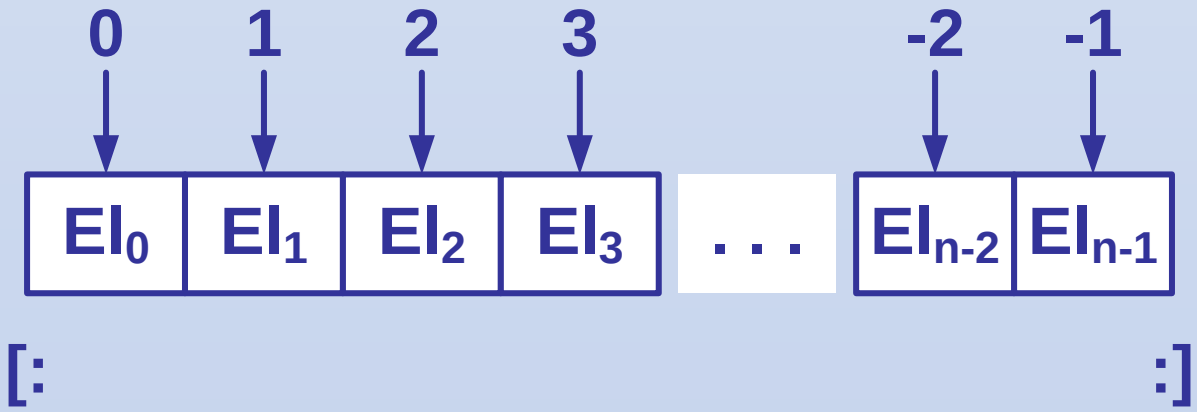
https://habr.com/ru/post/319164/

КОЛЕКЦІЇ


КОЛЕКЦІЇ
sequences
mappings
ПОСЛІДОВНОСТІ
ВІДОБРАЖЕННЯ
immutable
mutable
dict
НЕЗМІННІ
ЗМІННІ
СЛОВНИК
string
list
РЯДОК
СПИСОК
РЯДКИ байтів (3.0)
МАСИВ байтів (3.0)
КОРТЕЖ
tuple
sets
МНОЖИНИ
set
МНОЖИНА
frozenset
Фіксована МНОЖИНА

КОЛЕКЦІЇ. Властивості

Індексованість **- кожен елемент колекції має свій порядковий номер - індекс. Це дозволяє звертатися до елементу по його порядковому індексу, проводити слайсінг («нарізку») - брати частину колекції вибираючи виходячи з їх індексу.**



КОЛЕКЦІЇ. Властивості

Змінність колекції **- дозволяє додавати в колекцію нових членів або видаляти їх після створення колекції.**

Незмінні (***числа, рядки, кортежі, фіксовані множини***): не підтримують можливість безпосередньої зміни значення об'єкта, однак завжди можна створити нові об'єкти за допомогою виразів і привласнювати їх необхідним змінним.

Змінні (***списки, словники, множини***): завжди можуть змінюватися безпосередньо, за допомогою операцій, які не створюють нові об'єкти. Змінні об'єкти можуть бути скопійовані, але вони підтримують і можливість безпосереднього зміни.

КОЛЕКЦІЇ. Властивості

Унікальність **- кожен елемент колекції може зустрічатися в ній тільки один раз. Це породжує вимогу незмінності використовуваних типів даних для кожного елемента, наприклад, таким елементом не може бути список.**

КОЛЕКЦІЇ. Властивості

Тип	Змінність	Індексованість	Унікальність	Створення
list	+	+	-	*[ ]* *list()*
tuple	-	+	-	*()* *tuple()*
string	-	+	-	*‘ ‘* *“ “*
set	+	-	+	*{el1, el2, }* *set()*
frosenset	-	-	+	*frosenset()*
dict	+ елементи - ключі + значення	-	+ елементи + ключі - значення	*{}* *{key: value}* *dict()*

КОЛЕКЦІЇ. Стандартні функції

	Функція	Дія
1	**type()**	Тип колекції
2	**print()**	Друкування елементів колекції
3	**len()**	Кількість членів колекції
4	**X in S**	Перевірка входження елемента **X** в колекцію **S**
5	**min()**	Пошук мінімального елемента
6	**max()**	Пошук максимального елемента
7	**sum()**	Сума елементів (числових)

КОЛЕКЦІЇ. Стандартні методи

	.count ()	.index()	.copy()	.clear()
list	+	+	- (<3.3) + (>=3.3)	- (<3.3) + (>=3.3)
tuple	+	+	-	-
string	+	+	-	-
set	-	-	+	+
frosenset	-	-	+	-
dict	-	-	+	+

КОЛЕКЦІЇ. Стандартні методи

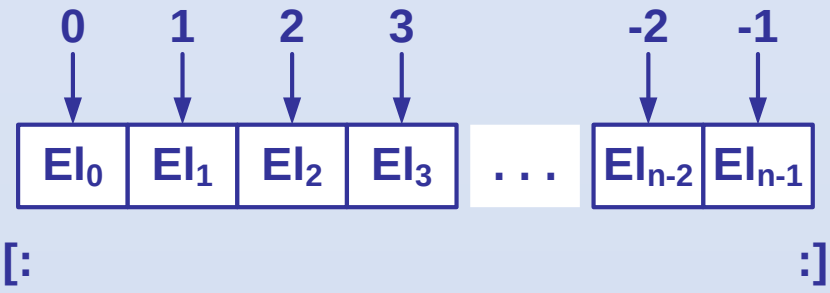
.count () - метод підрахунку певних елементів для неунікальний колекцій (рядок, список, кортеж), повертає скільки разів елемент зустрічається в колекції.

.index () - повертає мінімальний індекс переданого елемента для індексованих колекцій (рядок, список, кортеж)

.copy () - метод повертає неглибоку копію колекції (список, словник, обидва типи множини).

.clear () - метод змінюваних колекцій (список, словник, множина), що видаляє з колекції все елементи і перетворює її в порожню колекцію.

ПОСЛІДОВНОСТІ. Індексування



Для всіх індексованих колекцій можна отримати значення елемента по його індексу в квадратних дужках.

! можна задавати **негативний індекс** (зворотній порядок індексації).

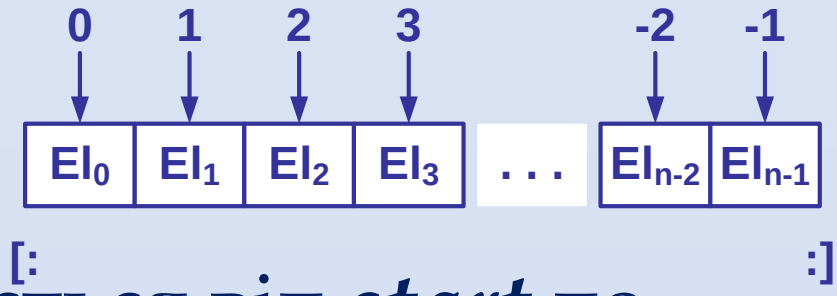
При завданні негативного індексу, останній елемент має індекс -1, передостанній -2 і так далі до першого елемента індекс якого дорівнює значенню довжини колекції з негативним знаком, тобто

-len (mycollection).

ПОСЛІДОВНОСТІ. Зрізи (slice)

Slice index

[*start* : *stop* : *step*]



Індекс елемента змінюється від ***start*** до ***stop-1*** включно з кроком ***step***

Варіанти

[: *stop* : *step*] – від 0 до ***stop-1*** з кроком ***step***

[*start* : : *step*] – від ***start*** до ***len()-1*** з кроком ***step***

[: *stop*] – від 0 до ***stop-1*** з кроком ***1***

[*start* :] – від ***start*** до ***len()-1*** з кроком ***1***

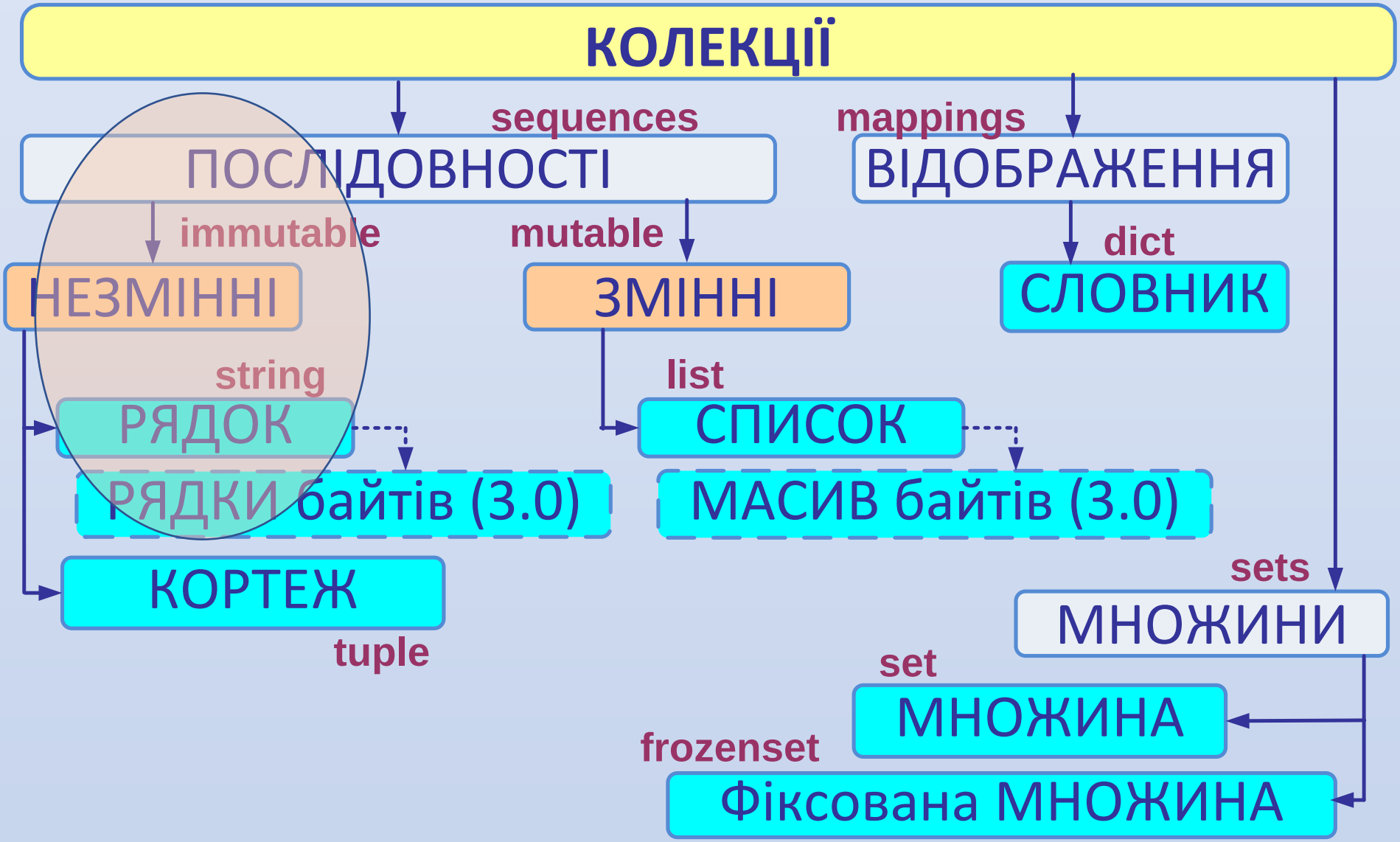
[:] – вся послідовність

ПРИМІТКА : [stop] не включається в результат

ПОСЛІДОВНОСТІ. Зрізи. Приклади

List →	A	B	C	D	E	F	G	Result
	0 (-7)	1 (-6)	2 (-5)	3 (-4)	4 (-3)	5 (-2)	6 (-1)	
[:] →	+	+	+	+	+	+	+	ABCDEFG
[::-1] ←	+	+	+	+	+	+	+	GFEDCBA
[::2] →	+		+		+		+	ACEG
[1::2] →		+		+		+		BDF
[:1]	+							A
[-1:]							+	G
[3:4]				+				D
[-3:] →					+	+	+	EFG
[-3:1:-1] ←			+	+	+			EDC
[2:5] →			+	+	+			CDE

РЯДКИ


КОЛЕКЦІЇ
sequences
ПОСЛІДОВНОСТІ
mappings
ВІДОБРАЖЕННЯ
immutable
mutable
dict
НЕЗМІННІ
ЗМІННІ
СЛОВНИК
string
list
РЯДОК
СПИСОК
РЯДКИ байтів (3.0)
МАСИВ байтів (3.0)
КОРТЕЖ
tuple
sets
МНОЖИНИ
set
МНОЖИНА
frozenset
Фіксована МНОЖИНА

РЯДКИ

Рядок = колекція → незмінна послідовність одно символьних рядків

Рядок як послідовність підтримує порядок розміщення елементів, які вона містить (символи в Unicode !), зліва направо: елементи зберігаються і витягуються виходячи з їх позиції в послідовності.

Тип	Змінність	Індексованість	Унікальність	Створення
string	-	+	-	‘ ‘ “ “

РЯДКИ. Базові операції

ЛІТЕРАЛИ	
S=‘‘ S=“”	Пустий рядок (апострофи, лапки)
S=“spam’s”	Рядок в лапках
S=‘s\np\ta\x00m	Екранований рядок
Block = “””... “””	Блок (потроєні лапки)
S= r’\temp\spam’	Неформатований рядок
S= b’spam’	Рядок байтів (>=3.0)
S= u’spam’	Рядок в unicode

Екранований рядок – екрановані пари

СЛЕШ СИВОЛ => СЕЦСИМВОЛ (!!! Один байт)

\n - символ new line (ASCII cod = 10)

\t – символ tabulation

\r – повернення каретки

РЯДКИ. Базові операції

Конкатенація	+	‘ab’+”123”->’ab123’
Дублювання	*	‘abc’*2 -> ‘abcabc’
Розмір рядка	len()	Кількість символів
Вибірка за індексом	[i]	*i*-й символ рядку
Зріз	[start: stop: step]	частина рядку

РЯДКИ. Функції

	Функція	Дія
1	print()	Друкування рядку
2	len()	Кількість символів в рядку
3	X in S	Перевірка входження елемента підрядку **X** в рядок **S**
4	min()	Пошук мінімального символу
5	max()	Пошук максимального символу
6		

РЯДКИ. Методи (>20)

	ВЕРТАЄ
S.lower()	Копію рядка, всі символи малі (lowercase)
S.upper()	Копію рядка, всі символи великі (uppercase)
S.swapcase()	Копію рядка, символи змінені (великі<-> малі)
S.split (sep, maxsplit)	Кількість слів в рядку. Роздільник **sep**
…	
S.is… () **S.isalpha()** **S.isdecimal()** **S.islower()** **. . .**	**True** коли виконується, **False** інакше Усі символи рядка є букви Усі символи рядка є десяткові цифри Усі символи рядка є в нижньому регістрі
…	
S.find (sub[,start[,end]])	Найменший індекс в рядку, де підрядок **sub** знаходиться в зрізі [**start:end**]
S.format(*args, *kwargs)	Форматування рядка

Форматування рядків (1)

Вирази форматування – базується на моделі функції ***printf*** мови **C.**

Оператор **%** - дає можливість множинної підстановки рядків. Використання:

1. Зліва від оператора% вказати рядок формату, що містить один або більше специфікаторів формату, кожен з яких починається з символу% (наприклад,% d).

2. Праворуч від оператора% вказати об'єкт (або об'єкти, у вигляді кортежу), значення якого має бути підставлено на місце специфікатору (або специфікаторів) в лівій частині виразу.

Форматування рядків (1)

%[(*name*)][*flags*][*width*][.*precision*]*code*

name - ключ

flags - список признаків (+, -, 0, …)

width , *precision* – кількість символів

code	
s	Рядок
c	Символ
d	Десяткове (ціле) число
i	Ціле число
f	Дійсне число
e	Дійсне в експоненціальній формі

Форматування рядків (1)

Приклади:

Num = 1234

Res= ‘integers: … %d…%-6d…%06d’ %(Num,Num,Num)

print (Res)

Out:

integers: …1234…1234…001234

fln = 98.23456789

print ('%e %f %.4f' %(fln,fln,fln))

Форматування рядків (2)

Метод форматування **s.format()** починаючи з версії 3.0

Ідея: записується рядок-шаблон **s**, який викликає метод формат **.format(),** в який передаються відповідні позиційні та іменовані аргументи

В рядку-шаблоні {0} {1} {2} …. /позиційні/, {nam1}: {name2} : … /іменовані/ змінні приймають відповідні аргументи методу.

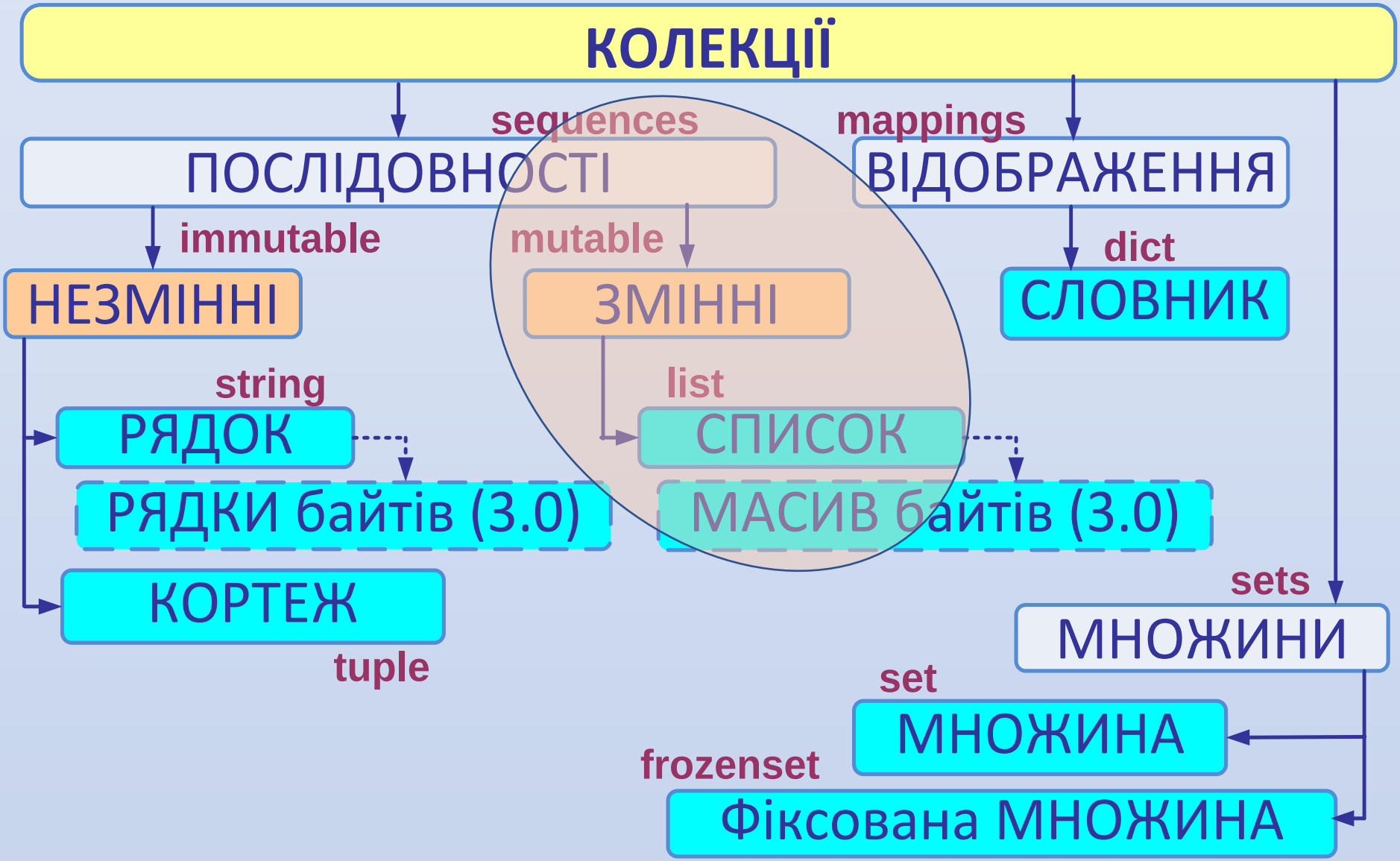
Приклад:

template = '{mt}: {0} - {fd}'

print(*template.format('ham', mt='spam',fd='123')*)

spam:ham - 123

СПИСКИ


КОЛЕКЦІЇ
sequences
mappings
ПОСЛІДОВНОСТІ
ВІДОБРАЖЕННЯ
immutable
mutable
dict
НЕЗМІННІ
ЗМІННІ
СЛОВНИК
string
list
РЯДОК
СПИСОК
РЯДКИ байтів (3.0)
МАСИВ байтів (3.0)
КОРТЕЖ
tuple
sets
МНОЖИНИ
set
МНОЖИНА
frozenset
Фіксована МНОЖИНА

СПИСКИ

Список = → упорядкована колекція об'єктів довільного типу

Список як послідовність підтримує порядок розміщення елементів, які вона містить. Доступ до елементів за зміщенням (індексом).

Змінна кількість елементів.

Довільне число рівнів вкладеності.

Тип	Змінність	Індексованість	Унікальність	Створення
list	+	+	-	*[ ]* *list()*

СПИСКИ. Створення

Створення списку

	Дія
L=[]	Пустий список
L=[5, 6, 7, 8]	Чотири елементи з індексами 0..3
L=[‘abc’,[‘def’,’ghi’]]	Вкладені списки
L=list(range(-4,4))	Створення списку
L=list(‘abcdef’)	Створення списку

СПИСКИ. Базові операції/функції

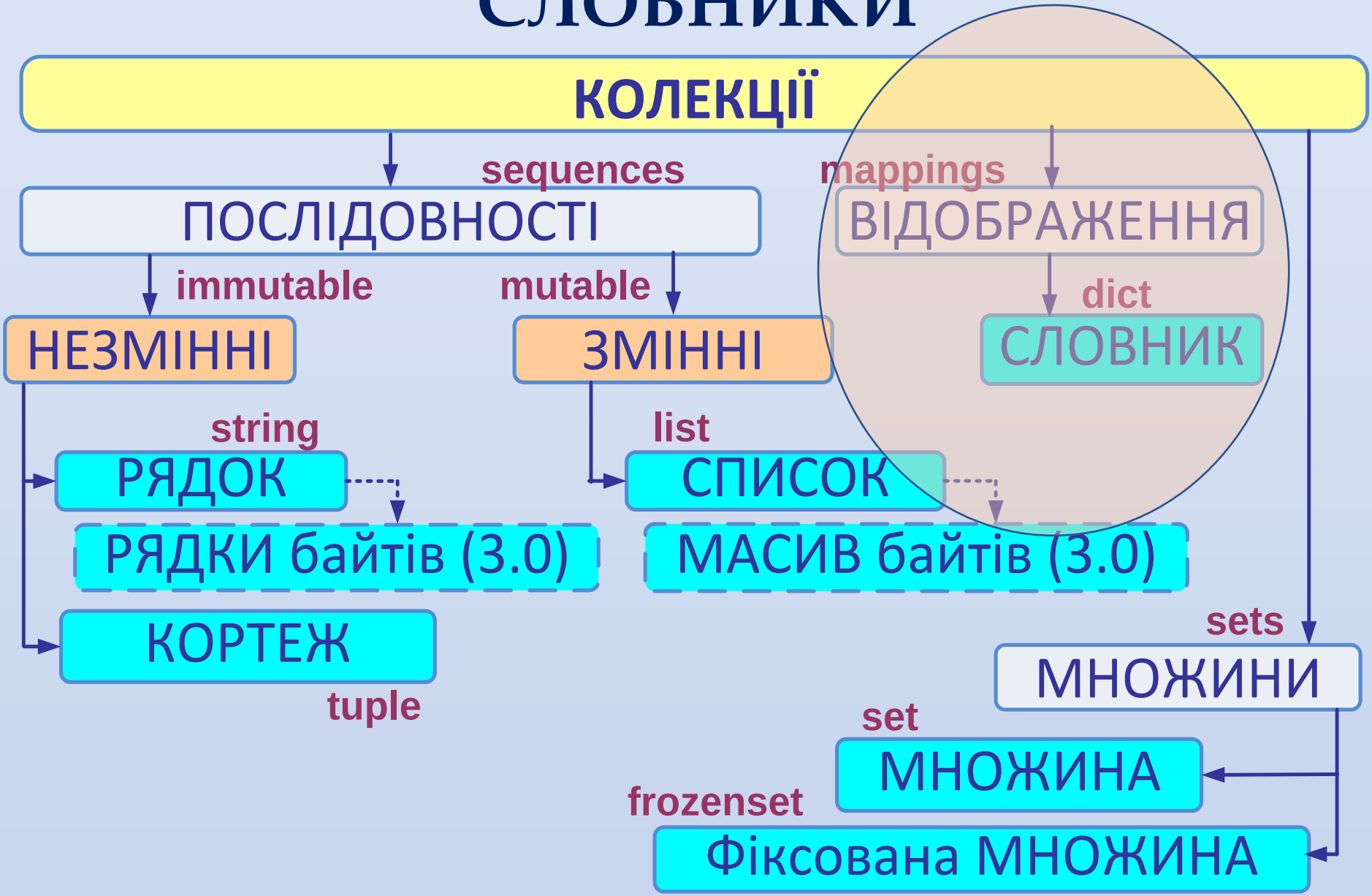
Операція		
Конкатенація	L1+L2	
Дублювання	L * N	N-разів повторення
Вибірка за індексом	L[i]	*i*-й об'єкт списку
Зріз	L[start: stop: step]	Новий список = зрізу

Функція	Дія
print(L)	*Друкування елементів списку*
len(L)	*Кількість об'єктів в списку*
X **in** L	*Перевірка входження об'єкту X в список S*
min(L)	*Пошук мінімального елемента*
max(L)	*Пошук максимального елемента*
sum(L)	*Сума елементів (числових)*

СПИСКИ. Методи

Метод	Дія
L[i] = … L[sr:st:sp] = L = [generator]	*Присвоєння за індексом* *L[5]=34* *L[1:3:1]= 34,15,18*
L.**append**() L.**extend**() L.**insert**()	*Додавання об'єктів до списку*
del L[k] L.**pop**() L.**remove**() L[sr:st:sp] =[]	*Зменшення об'єктів в списку*
L.**sort** ()	*Сортування*
L.**reverse**()	*Зміна порядку на зворотній*

СЛОВНИКИ


КОЛЕКЦІЇ
sequences
ПОСЛІДОВНОСТІ
mappings
ВІДОБРАЖЕННЯ
immutable
mutable
dict
НЕЗМІННІ
ЗМІННІ
СЛОВНИК
string
list
РЯДОК
СПИСОК
РЯДКИ байтів (3.0)
МАСИВ байтів (3.0)
КОРТЕЖ
tuple
sets
МНОЖИНИ
set
МНОЖИНА
frozenset
Фіксована МНОЖИНА

СЛОВНИКИ

Словник = → Неупорядкована змінна колекція об'єктів довільного типу

Словник забезпечує доступ до елементів за ключем.

В словнику КЛЮЧ це індекс!!!

Змінна кількість елементів.

Довільне число рівнів вкладеності.

Тип	Змінність	Індексованість	Унікальність	Створення
dict	+ елементи - ключі + значення	-	+ елементи + ключі - значення	*{}* *{key: value}* *dict()*

СЛОВНИКИ. Створення

Створення словнику

	Дія
D={}	Пустий словник
D={5:’as’, 6:’is’, 7:’if’}	Словник з трьох елементів
D={’as’:5, ’is’:25, ’if’:’OK’}	Словник з трьох елементів
D=dict(name=‘Piter’, age=35)	Функція створення словнику
D=dict(zip(kyelist, vallist))	Функція створення словнику
D={5:’as’,6:{’as’:5, ’is’:25} 7:’if’}	Словник з вкладеним словником

СЛОВНИКИ. Базові операції/функції

Операція		
Вибірка ключем	D[key]	*key*-й об'єкт словнику
Вибірка ключем	D[6][‘is’]	Вибірка з вбудованого словника

Функція	Дія
print(D)	*Друкування елементів словнику*
len(D)	*Кількість об'єктів в словнику*
key **in** D	*Перевірка на входження об'єкту з ключем* ***key*** *в словник* D

СЛОВНИКИ. Методи

Метод	Дія
D[key] =	*Додавання ключа + значення*
D.**items**() D.**keys**() D.**values**()	*Список ключів та значень* *Список ключів* *Список значень*
D.**copy**()	*Копіювання словника*
D.**get**(key[,default])	*Витяг за ключем, якщо ключа немає вертається* ***default*** *or* ***None***
D.**update**(D2)	*Оновлення словника додавання пар з D2*
D.**pop**(key)	*Повернення значення та видалення*

Контрольні запитання

- Надайте визначення колекції в мові Python, наведіть перелік вбудованих типів колекцій, вкажіть базові властивості колекцій.
- Надайте визначення **зрізу** для послідовностей, наведіть приклади формування зрізів.
- Надайте перелік основних **операцій із рядками**, вкажіть їх призначення та наведіть відповідні приклади.
- Надайте перелік основних **функцій** об'єктів типу рядок, вкажіть їх призначення та наведіть відповідні приклади.
- Надайте перелік основних **методів** об'єктів типу рядок, вкажіть їх призначення та наведіть відповідні приклади.
- Вкажіть способи форматування рядків, наведіть відповідні приклади.

Контрольні запитання

- Надайте визначення **списку** в мові Python, вкажіть властивості списку, варіанти створення списку. Наведіть приклади.
- Надайте перелік основних **операцій** із **списками**, вкажіть їх призначення та наведіть відповідні приклади.
- Надайте перелік основних **функцій** об'єктів типу **список**, вкажіть їх призначення та наведіть відповідні приклади.
- Надайте перелік основних **методів** об'єктів типу **список**, вкажіть їх призначення та наведіть відповідні приклади.

Контрольні запитання

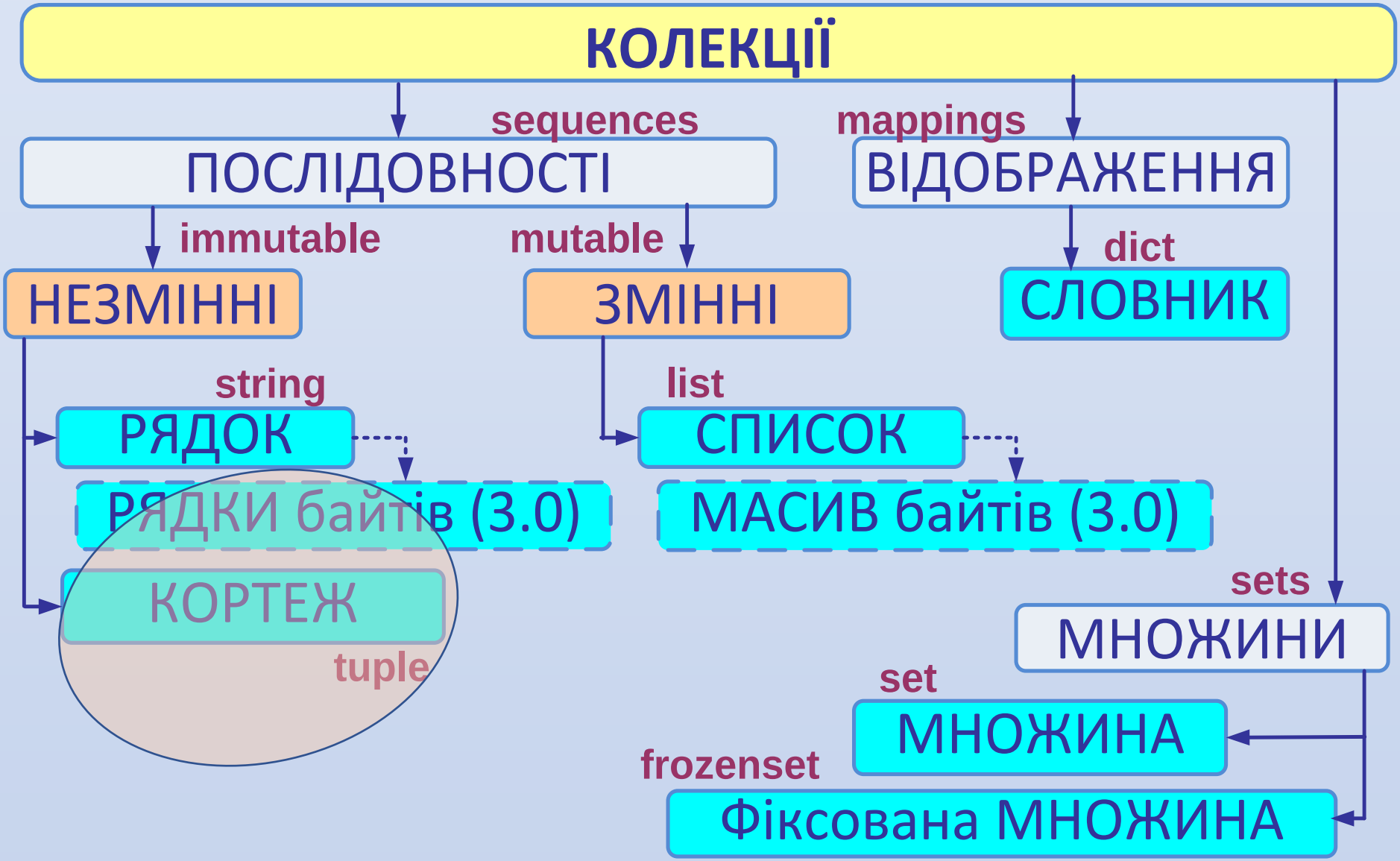
- Надайте визначення **словника** в мові Python, вкажіть властивості словника, варіанти створення словника. Наведіть приклади.
- Надайте перелік основних операцій із **словниками**, вкажіть їх призначення та наведіть відповідні приклади.
- Надайте перелік основних **функцій** об'єктів типу **словник**, вкажіть їх призначення та наведіть відповідні приклади.
- Надайте перелік основних **методів** об'єктів типу **словник**, вкажіть їх призначення та наведіть відповідні приклади.

The END
Лекція 03

ЛЕКЦІЯ 04. PYTHON #3

1. Кортеж (tuple)
2. Множина (set)

КОРТЕЖИ


КОЛЕКЦІЇ
sequences
mappings
ПОСЛІДОВНОСТІ
ВІДОБРАЖЕННЯ
immutable
mutable
dict
НЕЗМІННІ
ЗМІННІ
СЛОВНИК
string
list
РЯДОК
СПИСОК
РЯДКИ байтів (3.0)
МАСИВ байтів (3.0)
КОРТЕЖ
tuple
sets
МНОЖИНИ
set
МНОЖИНА
frozenset
Фіксована МНОЖИНА

КОРТЕЖ

Кортеж = → упорядкована незмінна колекція об'єктів довільного типу

Кортеж як послідовність підтримує порядок розміщення елементів, які вона містить. Доступ до елементів за зміщенням (індексом).

Незмінна кількість елементів.

Довільне число рівнів вкладеності.

Тип	Змінність	Індексованість	Унікальність	Створення
tuple	-	+	-	*()* *tuple()*

Кортеж – масив указників на елементі

КОРТЕЖ. Створення

Створення кортежу

	Дія
T=()	Пустий кортеж
T=(5, ‘6’, 7, ‘8’)	Чотири елементи з індексами 0..3
T=(‘abc’,(‘def’,ghi))	Вкладений кортеж
T=tuple(‘abcdef’)	Створення кортежу
T=tuple(range(…))	Створення кортежу

КОРТЕЖ . Базові операції/функції

Операція		
Конкатенація	T1+T2	
Дублювання	T * N	N-разів повторення
Вибірка за індексом	T[i]	*i*-й об’єкт кортежу
Зріз	T[start: stop: step]	Новий кортеж = зрізу

Функція	Дія
print(T)	*Друкування елементів кортежу T*
len(T)	*Кількість об'єктів в кортежу T*
X **in** T	*Перевірка входження об'єкту X в кортеж T*
min(T)	*Пошук мінімального елемента кортежу T*
max(T)	*Пошук максимального елемента кортежу T*
sum(T)	*Сума елементів (числових) кортежу T*

КОРТЕЖ. Методи

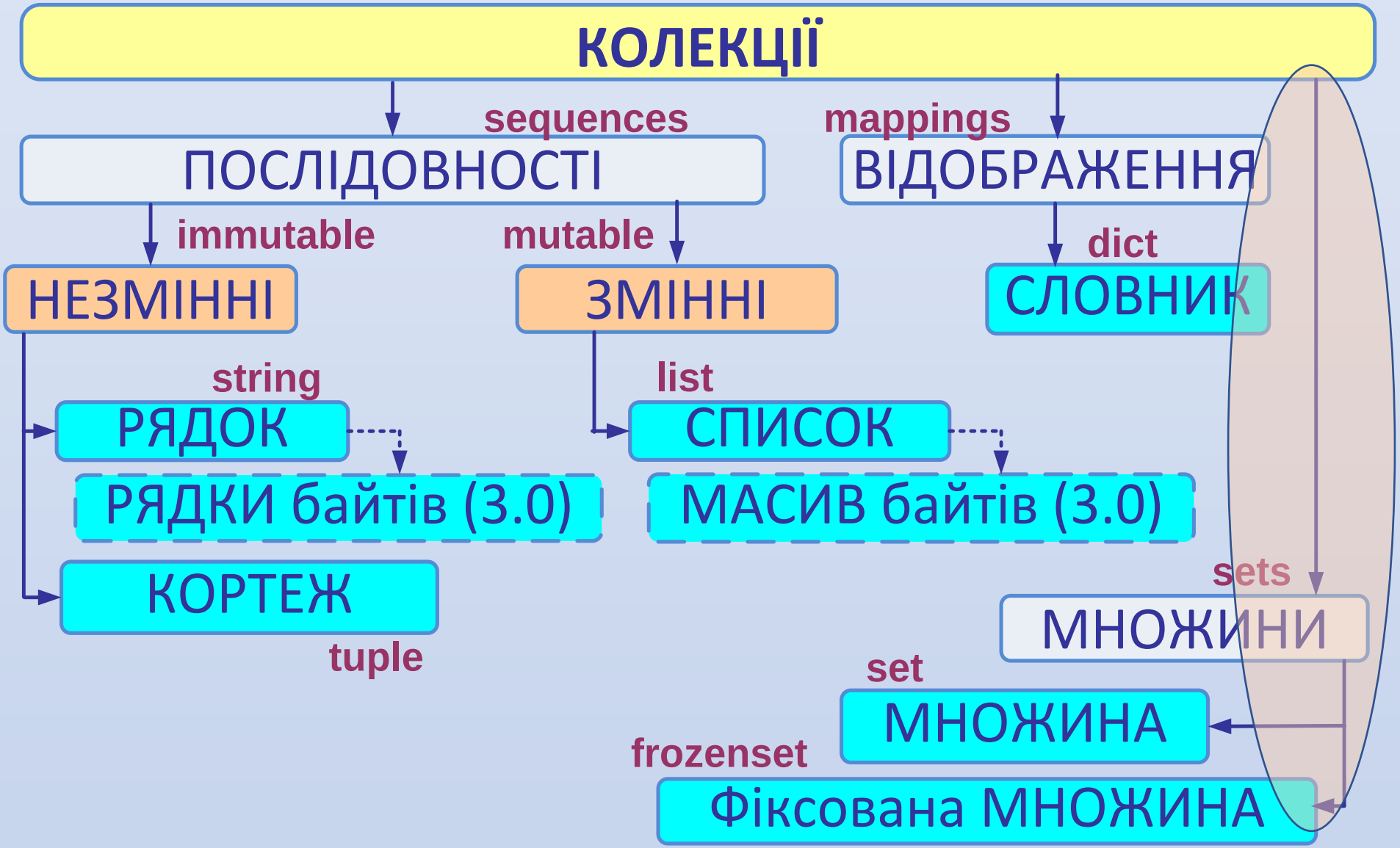
Метод	Дія
T.**index(EL)**	*Індекс елементу* EL *в кортежі* T
T.**count (EL)**	*Кількість елементів* EL *в кортежі* T

Приклади

- Сортування кортежу
- Додавання елементу
- Зміна елементу

ДИВИСЬ EXAMPL_LEC_04_PYTHON_03_Tuple.ipynb

МНОЖИНА


КОЛЕКЦІЇ
sequences
mappings
ПОСЛІДОВНОСТІ
ВІДОБРАЖЕННЯ
immutable
mutable
dict
НЕЗМІННІ
ЗМІННІ
СЛОВНИК
string
list
РЯДОК
СПИСОК
РЯДКИ байтів (3.0)
МАСИВ байтів (3.0)
КОРТЕЖ
tuple
sets
МНОЖИНИ
set
МНОЖИНА
frozenset
Фіксована МНОЖИНА

МНОЖИНА (> 3.0)

Множина → НЕупорядкована змінна - set (незмінна - frosenset)) колекція унікальних об'єктів довільного типу

Тип	Змінність	Індексованість	Унікальність	Створення
set	+	-	+	*{el1, el2, } set()*
frosenset	-	-	+	*frozenset()*

МНОЖИНА. Створення

Створення множини

	Дія
St={el1 , el2 , , }	Множина елементів
St=set('gsgjsdh')	Множина елементів
St=set([*iterable*])	Множина елементів (ітератор)
St=frozenset([*iterable*])	Фіксована множина елементів (ітератор)

https://docs.python.org/3/library/stdtypes.html?highlight=set#set

МНОЖИНА. Базові операції/функції

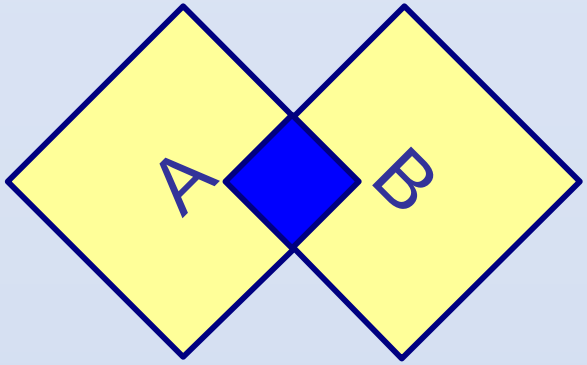
Функція	Дія
print(St)	*Друкування елементів словнику*
len(St)	*Кількість об'єктів в словнику*
el **in** St	*Перевірка на входження об'єкту з ключем* ***key*** *в словник* D
for el **in** St	

МНОЖИНА. Методи

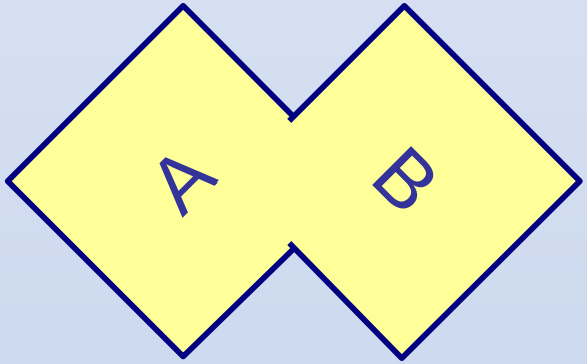
Метод	Дія
St.**copy**() Fst.**copy**()	*Вертає копію множини*
St.**clear**()	*Видалення всіх елементів множини*
St.**add**(el)	*Додає* **el** *до множини*
St.**discard**(el)	*Видаляє* **el,** *якщо він є*
St.**remove**(el)	*Видаляє* **el,** *якщо відсутній KeyError*
St.**pop**()	*Видаляє перший елемент та вертає його*

Метод	Дія
set_a.**isdisjoint**(set_b)	***Set_A*** *неперерізний з* ***Set_B ???***
set_a.**issubset**(set_b)	***Set_A*** *підмножена* ***Set_B ???***
set_a.**issuperset**(set_b)	***Set_A*** *надмножена* ***Set_B ???***

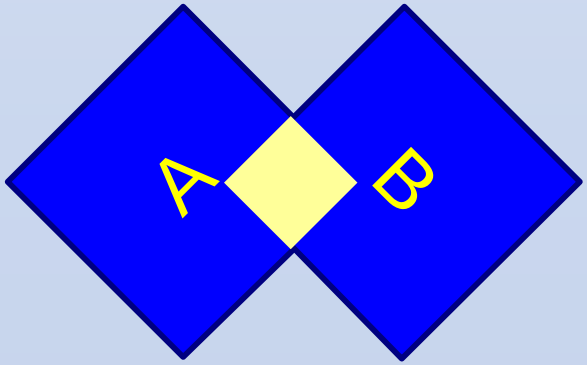
МНОЖИНИ. Методи



Метод
set_c=set_a.**intersection**(set_b) set_c=set_a **&** set_b

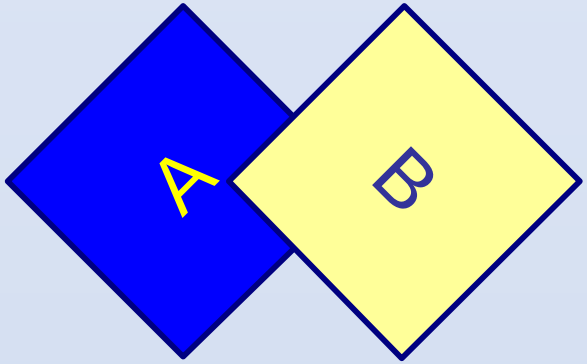


Метод
set_c=set_a.**union**(set_b) set_c=set_a **|**set_b

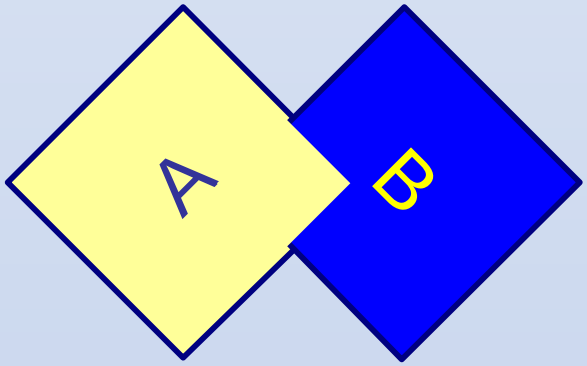


Метод
set_c=set_a.**symmetric_difference** (set_b) set_c=set_a **^** set_b

МНОЖИНИ. Методи



Метод
set_c=set_a.**difference**(set_b)



Метод
set_c=set_b.**difference**(set_a)

Контрольні запитання

- Надайте визначення **кортежу** в мові Python, вкажіть властивості кортежу, варіанти створення кортежу. Наведіть приклади.
- Надайте перелік основних операцій із **кортежами**, вкажіть їх призначення та наведіть відповідні приклади.
- Надайте перелік основних функцій об'єктів типу **кортеж**, вкажіть їх призначення та наведіть відповідні приклади.
- Надайте перелік основних методів об'єктів типу **кортеж**, вкажіть їх призначення та наведіть відповідні приклади.

Контрольні запитання

- Надайте визначення **множини** в мові Python, вкажіть властивості множини, варіанти створення множини. Наведіть приклади.
- Надайте перелік основних операцій із **множиною**, вкажіть їх призначення та наведіть відповідні приклади.
- Надайте перелік основних функцій об'єктів **множина**, вкажіть їх призначення та наведіть відповідні приклади.
- Надайте перелік основних методів об'єктів типу **множина**, вкажіть їх призначення та наведіть відповідні приклади.

The END
Лекція 04

ЛЕКЦІЯ 05. PYTHON #4

1. Файли в Python
2. Огляд інструкцій Python
3. Структура програми

ФАЙЛ

Файл – іменована область постійної пам'яті в комп'ютері, якою управляє операційна система.

Файл – вбудований тип об'єкту, що, забезпечує можливість доступу до цих областей пам'яті.

Об'єкт типу «файл» - має тільки методи та атрибути !

Файлові методи (операції)

Функція . Метод	Опис
Open ()	Створює об'єкт типу файл – «відкриває» файл
f.close()	Закриття відкритого файлу
f.Readable() f.Read(), f.readline(), f.readlines()	Перевірка можливості читання Читання
f.Seekable(), f.Seek()	Перевірка можливості зміни позиції
f.Tell()	Вертає поточну позицію
f.Writable(), f.write(), f.writelines()	Перевірка можливості запису Запись до файлу
f.next()	Вертає наступний рядок

https://www.tutorialspoint.com/python/file_methods.htm
https://www.w3schools.com/python/python_ref_file.asp

Файлові атрибути

Атрибут	Опис
f.closed	Індикатор поточного статусу файлу
f.encoding	Спосіб кодування/декодування файлу
f.mode	Режим відкриття файлу
f.name	Ім'я відкритого файлу
f.newlines	Універсальний режим завершення рядку ???

https://www.tutorialspoint.com/python/file_methods.htm
https://www.w3schools.com/python/python_ref_file.asp

Файлові методи (операції)

Вбудована функція	Опис
Open ()	**Створює об'єкт типу файл – «відкриває» файл**

Open(*file, mode, buffering, encoding, errors, newline, closed, opener*)

- ***file*** **–** path to file.
- ***mode*** *– режим (‘r’ - читання,’w’ - запис, ‘a’- додавання в кінець, ‘b’ – байтовий режим,‘t’-текст, ‘+’ – (r + w)). За замовчуванням ‘rt’ – читання тексту.*
- *buffering – управління буфером;*
- *encoding – режим кодування/декодування;*
- *errors – управління реакцією на помилки;*
- *newline – управління режимом «новий рядок»;*
- *closed– управління дескриптором файлу;*
- *opener – опис обробника помилки.*

Файлові методи (операції)

Вбудована функція	Опис
Open ()	**Створює об'єкт типу файл – «відкриває» файл**

Важливо: *'r', ...* текст – рядки типу **str** – виконується автоматичне кодування / декодування.

'b', – послідовність байтів – неяких змін.

***my_file* = open (*'my_text_file.txt', 'w')** – *відкриває файл для запису (тексту!)*

***my_file* = open (*'my_text_file.txt')** – *відкриває файл читання (тексту!)*

***my_file* = open (*'my_text_file.bin, 'wb')** – *відкриває файл для запису (послідовності байт)*

***my_file* = open (*'my_text_file.bin', 'rb')** – *відкриває файл читання (послідовності байт)*

Файлові методи (операції)

Метод	Опис
f.readable () **f.read ()**	**Зчитування файлу**

f.readable() – вертає **True,** коли файл відкрито для читання.

f.read(*size*) - зчитує деяку кількість даних і повертає їх у вигляді рядка (у текстовому режимі) або об'єкта байтів (у двійковому режимі); ***size*** – максимальна кількість даних, що зчитуються (необов'язковий числовий аргумент).

f.readline() - зчитує з файлу один рядок; символ нового рядка (\n) залишається в кінці рядка і опускається лише в останньому рядку файлу.

f.readlines() - зчитує всі рядки до списку.

Файлові методи (операції)

Метод	Опис
f.writeable () **f.write ()**	**Запис файлу**

f.writeable() – вертає **True,** коли файл відкрито для запису.

f.write(*string*) – записує вміст рядка до файлу і повертає кількість записаних символів.

f.writelines(*string*) – запис наступного елементу списку (рядок) до файлу (файл відкрито з *'wa'*)

!!! Інші типи об'єктів потрібно перетворити - або в рядок (в текстовому режимі), або в байт-об'єкт (у двійковому режимі) - перед їх записуванням.

Файлові методи (операції)

Метод	Опис
f.tell ()	**Вертає поточну позицію в файлі**
f.seek ()	**Змінює поточну позицію в файлі**

f.tell () – вертає ціле число, що дає поточне положення об'єкта файлу у файлі, представлене як кількість байтів від початку файлу (у двійковому режимі) і номер поточного символу в текстовому режимі.

f.seek (*offset, whence*) – нова позиція обчислюється від додавання ***offset*** до ***whence*** (опорної точки).

whence →

= 0 вимірює від початку файлу (за замовчуванням),

= 1 поточне положення файлу,

= 2 кінець файлу.

СТРУКТУРА PYTHON ПРОГРАМИ

Ієрархія програми →

Програма складається з пакетів та (або) модулів .

Пакет – Логічно завершена сукупність модулів .

Модуль – функціонально завершений фрагмент програми (оформлений як єдиний файл) - складається з інструкцій .

Інструкції складаються з виразів .

Вирази створюють та обробляють об'єкті .

СТРУКТУРА PYTHON ПРОГРАМИ

ІНСТРУКЦІЇ PYTHON

Інструкція	Дія	Приклад
import	Доступ до модуля	Import math
from	Доступ до атрибутів модуля	From sys import stdin
class	Створення об'єкту	Class sub(super): def …
try/except/ finally	Обробка виключень	Try: action () except : print()
raise	Створення виключення	
assert	Перевірки для налаштування	
with/as	Менеджер контексту	With open(‘file’) as myfile: proc(myfile)
del	Видалення посилань	

ІНСТРУКЦІЇ PYTHON

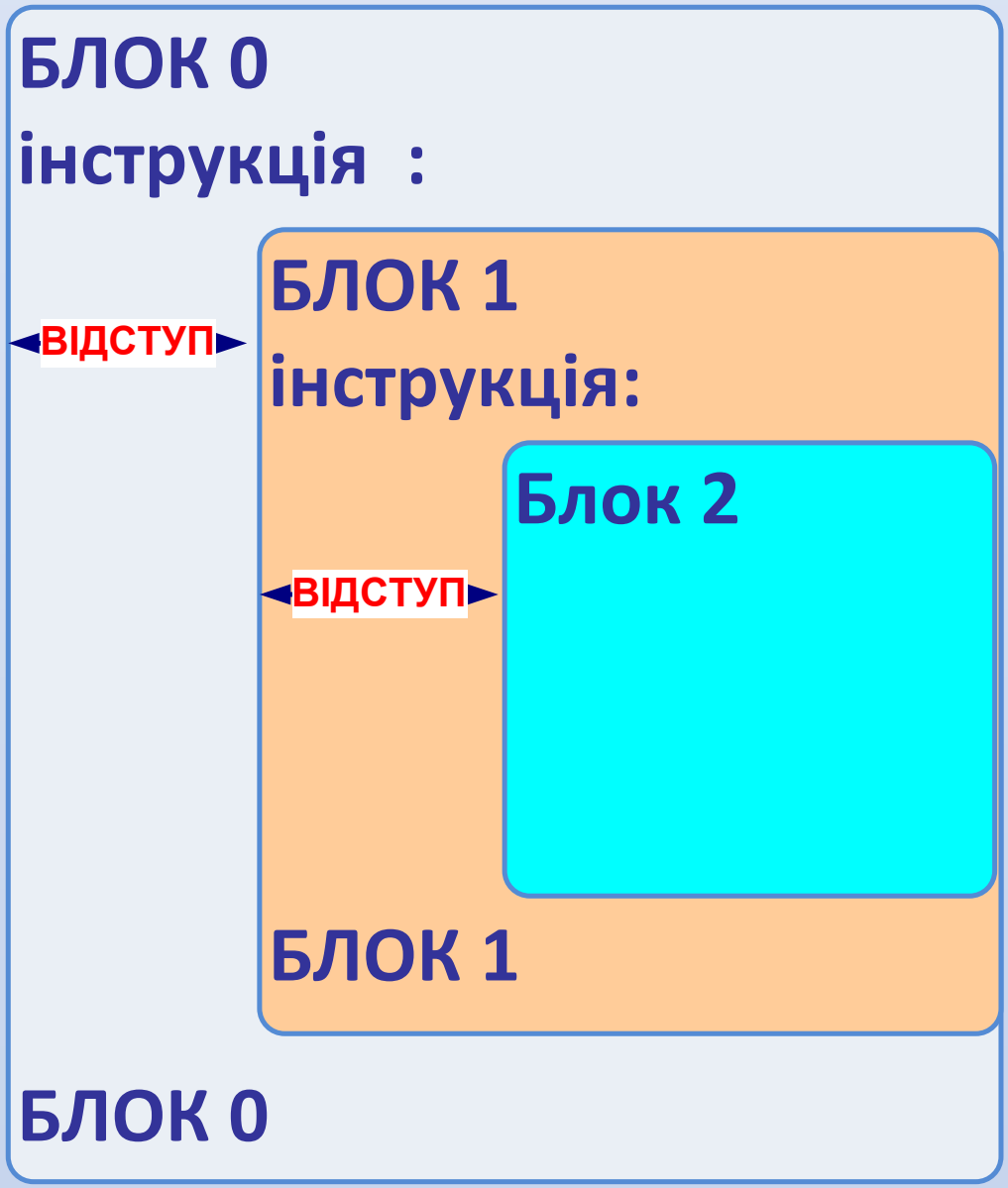
Інструкція	Дія	Приклад
Присвоювання	Створення посилань	A = “gglskfsj”
Виклики	Виклик функції	F = open()
if / elif / else	Вибір	If ‘z’ in text: print(text)
For / else	Перебір послідовності	for z in mlist: print(z)
while / else	Цикл загального призначення	while x>y: print(‘Qu’)
pass	Пуста інструкція	while True: pass

ІНСТРУКЦІЇ PYTHON

Інструкція	Дія	Приклад
break	Вихід з циклу	While True: if *cond* : break
continue	Перехід на початок циклу	While True: if *cond* : continue
def	Створення функцій та методів	Def foo (a): print (a)
return	Повернення результату	Def foo (a): return (a)
yield	Функції-генератори	Def gen (n): for i in n: yield (i)
global	Простір імен	
nolocal	Простір імен (3.0)	

СТРУКТУРА PYTHON ПРОГРАМИ

Вкладеність блоків регулюється відступами



Контрольні запитання

- Наведіть визначення файлу в мові Python. Надайте перелік основних методів об’єкту файл.
- Надайте визначення функції ***open()***, вкажіть призначення її параметрів та наведіть приклади використання.
- Надайте перелік основних атрибутів об’єкту файл та їх призначення.
- Надайте перелік методів **запису** до файлу, вкажіть їх призначення та наведіть приклади використання.
- Надайте перелік методів **читання** з файлу, вкажіть їх призначення та наведіть приклади використання.
- Надайте перелік методів **позиціювання** в файлі, вкажіть їх призначення та наведіть приклади використання.
- Наведіть перелік компонентів з яких складається програма на мові Python , вкажіть їх визначення.

The END
Лекція 05

ЛЕКЦІЯ 06. PYTHON #5

1. Функції
2. Поліморфізм
3. Область видимості
4. Зіставлення аргументів

ФУНКЦИІЇ

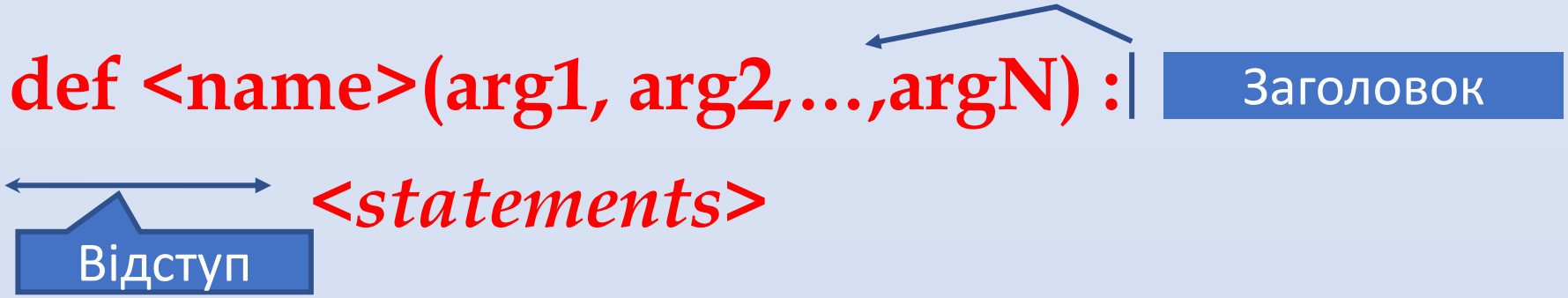
Функція → **базова програмна структура мови , що забезпечує багаторазове використання програмного коду і зменшує його надмірність.**

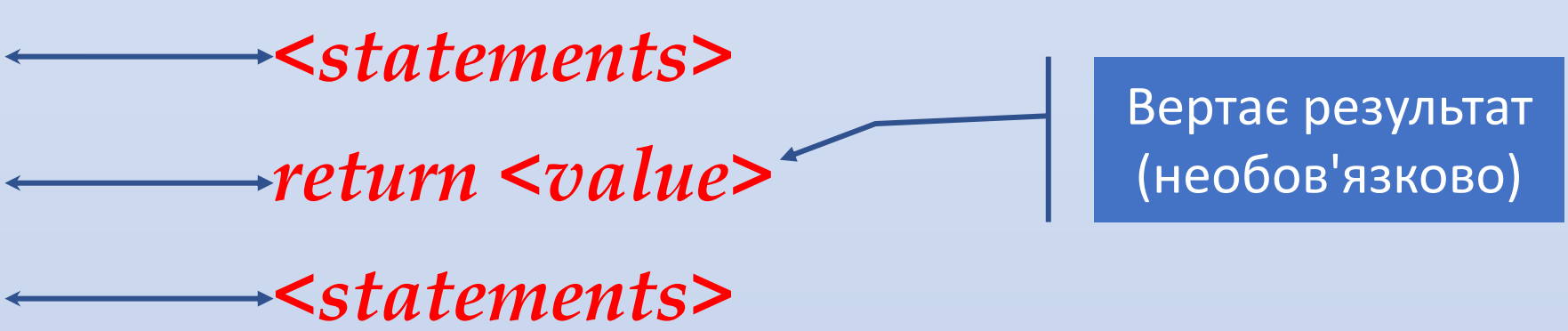
Функція → **засіб, що дозволяє групувати набори інструкцій так, що в програмі вони можуть запускатися неодноразово.**

ФУНКЦИІ

Інструкція	Приклад
def , return	Def myfunc(a,b): return a+b
Виклик	myfunc(1,2)
yield	Def sqr(x): for I in range (x): yield i**2
global	Def spb( ): global x; x = ‘new’
nonlocal	Def spb( ): nonlocal x; x = ‘new’
lambda	Func = [lambda x: x*2, lambda x: x**2]

СТВОРЕННЯ ФУНКЦИІЇ





def - ІНСТРУКЦІЯ, яка створює новий об'єкт типу функція і присвоює ім'я цьому об'єкту.

ПОЛІМОРІФЗМ

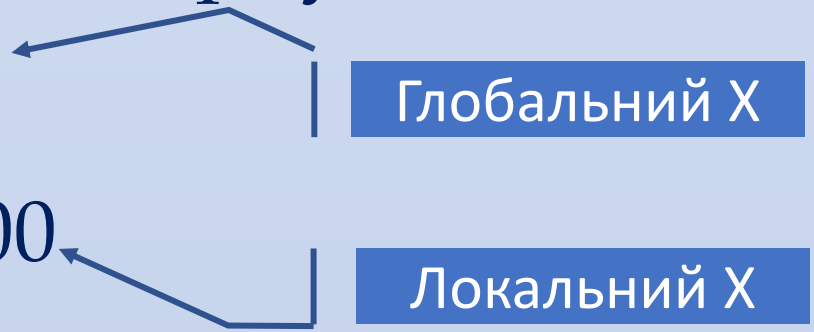
!!! ПОЛІМОРФІЗМ – сенс операції залежить висить від типів оброблюваних об'єктів. Python - це мова з динамічною типізацією, поліморфізм в ньому проявляється всюди.
Всі операції в мові Python є поліморфічні, поки об'єкти підтримують очікуваний інтерфейс (протокол), функція зможе обробляти їх.
Поліморфізм - це можливість обробки різних типів даних за допомогою "одне і тієї ж" функції, або методу.

ОБЛАСТЬ ВИДИМОСТІ

За замовчуванням всі імена, значення яких присвоюються всередині функції, асоціюються з простором імен цієї функції і ніяк інакше. Це **ЛОКАЛЬНІ** змінні.

- Імена, які визначаються всередині інструкції **def**, видно тільки програмного коду всередині інструкції **def**. До цих імен **неможна** звернутися за межами функції.

```
X = 100
def f():
    X=200
```



ОБЛАСТЬ ВИДИМОСТІ

- **LOCAL область** - присвоювання змінної виконується всередині інструкції ***def***, змінна є локальною для цієї функції.
- **NONLOCAL область -** присвоювання проводиться в межах охоплюючий інструкції ***def***, змінна є нелокальної для цієї функції.
- **GLOBAL область** присвоювання проводиться за межами всіх інструкцій ***def***, вона є глобальною для всього файлу.

Python - **лексична область видимості** - видимість змінної визначаються місцем розташування цієї змінної у вихідних текстах програми, а не місцем, звідки викликаються функції.

ІНСТРУКЦІЇ global, nonlocal

!!! Інструкції **global / nonlokcal** не оголошують тип або розмір змінної - вони оголошують **простір імен**.

Інструкція **global** дозволяє змінювати змінні, що знаходяться на верхньому рівні модуля, за межами інструкції **def**.

- *Глобальні імена - це імена, які визначені на верхньому рівні вмещающого модуля.*
- *Глобальні імена повинні оголошуватися, тільки якщо їм будуть присвоюватися значення всередині функцій.*
- *Звертатися до глобальних імен всередині функцій можна і без об'явлення їх глобальними.*

Інструкція **nonlocal** застосовується до імен, які перебувають в локальних областях видимості охоплюючий інструкцій **def**.

ПРАВИЛО LEGB

PYTHON Вбудована область видимості B

Зумевлені імена (open, range ….

МОДУЛЬ Глобальна область видимості G

Імена, визначені на верхнім рівні модуля або оголошені в інструкції **def** як глобальні

ФУНКЦІЯ non локальна область видимості

Імена, визначені в локаліній області видимості та в функції, яка охоплює E

ФУНКЦІЯ локальна області видимсті

Імена, визначені в тілі функції L

ОБЛАСТЬ ВИДИМОСТІ

```
X = 100
def func(Y):
    Z=X+Y
    return Z
print (func(1))
print (X)
```

```
101
100
```

```
X = 100
def func(Y):
    X=200
    Z=X+Y
    return Z
print (func(1))
print (X)
```

```
201
100
```

```
X = 100
def func(Y):
    global X
    X=200
    Z=X+Y
    return Z
print (func(1))
print (X)
```

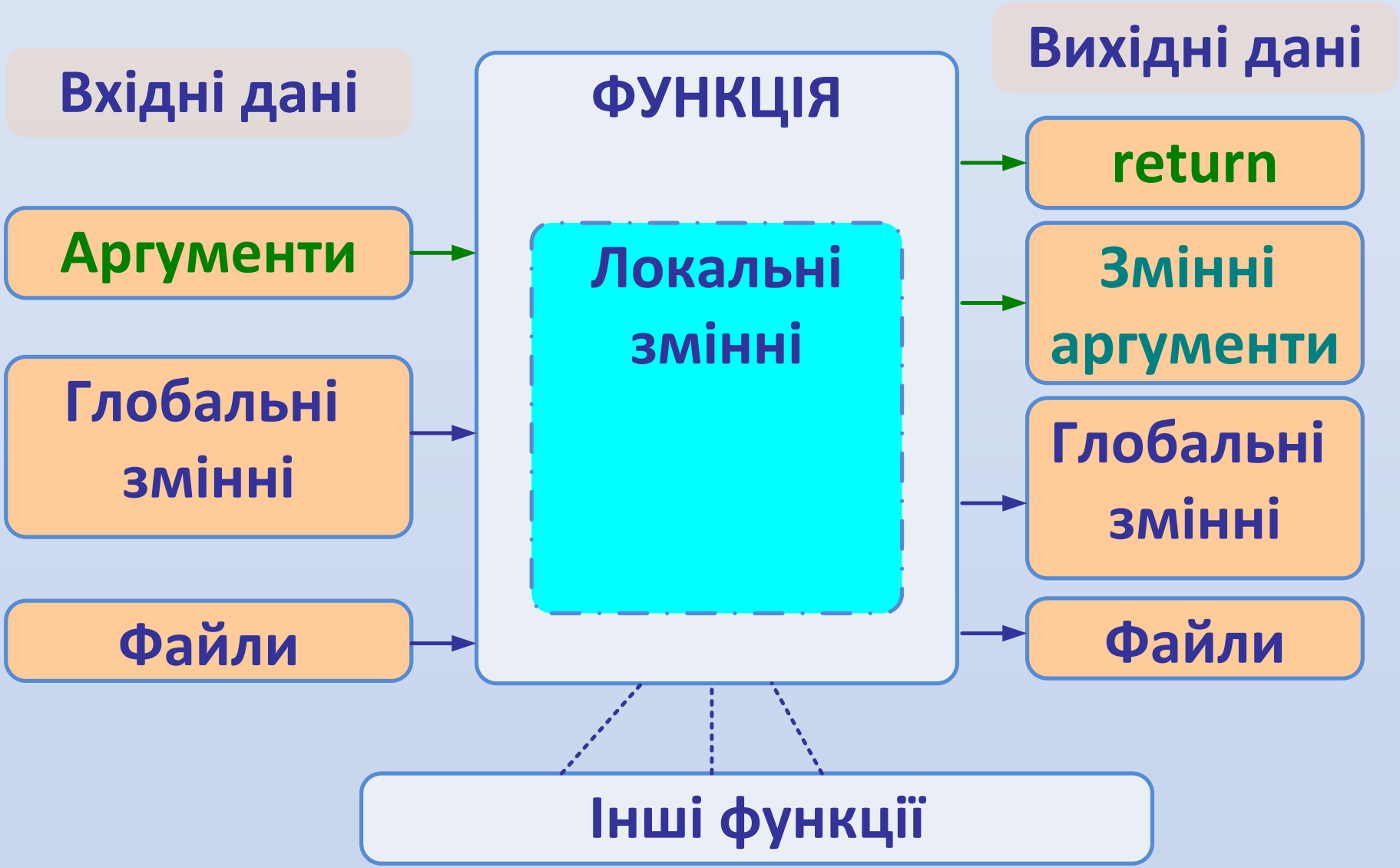
```
201
200
```

АРГУМЕНТИ

- **НЕЗМІННІ АРГУМЕНТИ** - передаються «за значенням» – передаються в вигляді **посилань** на об'єкти – (не в вигляді копій!). **Безпосередня зміни аргументу всередині функції НЕМОЖЛИВА.**

- **ЗМІННІ АРГУМЕНТИ -** передаються «за вказівником» – передаються в вигляді **вказівника . Допускається можливість безпосередньої зміни аргументу всередині функції.**

АРГУМЕНТИ


Вхідні дані
Аргументи
Глобальні змінні
Файли
ФУНКЦІЯ
Локальні змінні
Вихідні дані
return
Змінні аргументи
Глобальні змінні
Файли
Інші функції

АРГУМЕНТ за ЗАМОВЧУВАННЯМ

Дозволяють зробити окремі аргументи функції необов'язковими.
Якщо значення не передається, аргумент отримує значення за замовчуванням, яке визначено у заголовку функції

def <name>(arg1, arg2=<def_value>,…) :

<statements>

АРГУМЕНТИ. РЕКОМЕНДАЦІЇ

- Для передачі значень до функції використовуйте аргументи, для повернення результатів - інструкцію **return**.
- Не використовуйте глобальні змінні (! тільки якщо це дійсно необхідно).
- Не впливайте на змінювані аргументи, якщо модуль, що викликає, не передбачає цього.

ЗІСТАВЛЕННЯ АРГУМЕНТІВ

#Визначення функції

```
def foo(arg1, arg2,…,argN) :
    <statements>
```

#Виклик функції

```
foo(par_x, par_y,…, par_z)
```

Який параметр виклику відповідає аргументу визначення ?

ЗІСТАВЛЕННЯ АРГУМЕНТІВ

- **ПОЗИЦІЙНИ аргументи -** відповідність визначається за позицією аргументу у визначенні функції та її виклику.
- **ІМЕНОВАНІ аргументи –** дозволяють визначити відповідність за іменами, а не за позиціями аргументів.

ЗІСТАВЛЕННЯ АРГУМЕНТІВ

Опис	Виклик	
def func(name)	**func(value)**	За позицією або за ім'ям
def func (name=value)	**func (name=value)**	Значення аргументу за замовчуванням, якщо аргумент не передається функції
def func(*name)	**Func (*sequence)**	Довільне число аргументів за позицією (кортеж)
def func(name)**	**func(** dict)**	Довільне число аргументів за іменами (словник)
def func(*args, name)		≥3.0 Аргументи, що передаються тільки за іменами
def func(*, name-value)		

ЗІСТАВЛЕННЯ АРГУМЕНТІВ

У **заголовку** функції аргументи повинні вказуватися в наступному порядку:
будь-які звичайні аргументи (**name**),
- за якими можуть слідувати аргументи зі значеннями за замовчуванням (**name = value**),
-- за якими слідують аргументи в формі ***args** (або * в 3.0), якщо є ,
--- за якими можуть слідувати будь-які імена або пари **name = value** аргументів, які передаються тільки по імені (в 3.0),
---- за якими можуть слідувати аргументи в формі ****kwargs** (останні!).

ЗІСТАВЛЕННЯ АРГУМЕНТІВ

У **виклику** функції аргументи повинні вказуватися в наступному порядку: будь-які позиційні аргументи (**значення**), за якими можуть слідувати будь-які іменовані аргументи (**name = value**) і аргументи у формі ***sequence**, за якими можуть слідувати аргументи в формі ****dict** (остання, відповідає ****kwargs** заголовку).

ЗІСТАВЛЕННЯ АРГУМЕНТІВ

Дії інтерпретатору:

1. Сопоставлення неіменованих аргументів за позиціями.
2. Сопоставлення іменованих аргументів за іменами
3. Сопоставлення додаткових неіменованих аргументів з кортежем ***args***.
4. Сопоставлення додаткових іменованих аргументів з словником ****kwargs***.
5. Сопоствалення значень за замовчуванням з відсутніми іменованими

Контрольні питання

- Наведіть визначення функції в мові Python. Надайте опис інструкції створення функції. вкажіть призначення її компонентів.
- Наведіть перелік областей видимості для змінних функції. Надайте інструкції керування областями видимості для змінних функції. Наведіть приклади використання.
- Наведіть визначення аргументів за замовчуванням, наведіть приклади використання.
- Наведіть правила зіставлення аргументів функції у її описі та її виклику.

The END
Лекція 06

ЛЕКЦІЯ 07. PYTHON #6

1. **Рекурсивні функції**
2. **Непрямі виклики**
3. **Інтроспекція**
4. **Анотації функцій**
5. **Анонімні функції (*lambda*)**
6. **Відображення на послідовність: (*map*)**

РЕКУРСИВНА ФУНКЦІЯ

Визначено деяке x_0 .
Обчислення x_n -будується як x_i =F(x_{i-1}), i=1,2,… , n.
Кожне обчислення F(x) – *ітерація, i – номер ітерації,* процес – *ітеративний*.

З іншої сторони : x_n =F(F(F(… F(x_0))) – *рекурентний* процес обчислення x_n .

Рекурсія → **– метод визначення об'єкту через раніш визначені об'єкти, серед яких сам об'єкт.**

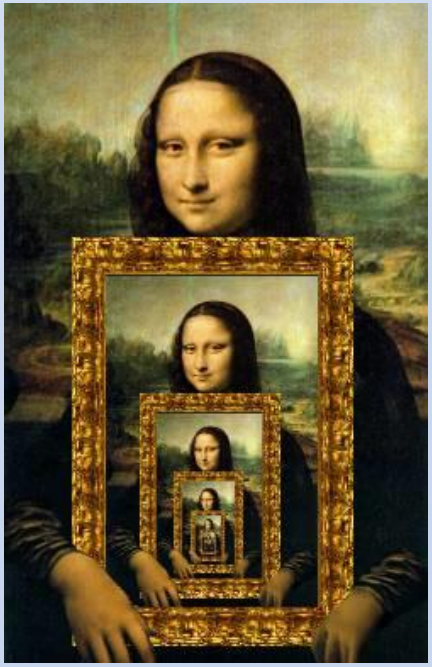
Ітерація – багаторазове повторення.
Рекурсія – багаторазове звернення.

РЕКУРСИВНА ФУНКЦІЯ

Теорема → ***рекурсивне*** **обчислення завжди можна перетворити в *ітераційне* обчислення (що використовує цикли). І навпаки, будь-яке *ітераційне* обчислення, що припускає використання циклів, можна реалізувати як *рекурсивне*.**

Рекурсія VS Ітерація →
предмет багаторічних суперечок програмістів

	Рекурсія	Ітерація
Запис	Компактний	**НЕ** компактний
Час	Повільніше	Швидше
Пам'ять	Більше	Менше



РЕКУРСИВНА ФУНКЦІЯ

Рекурсія → – метод визначення об'єкту через раніш визначені об'єкти, серед яких сам об'єкт.

Рекурсивна функція → — це така функція, серед виконуваних інструкція, якою є оператор виклику самої цієї функції.

```
def <name>(arg1, arg2,…,argN) :
        <statements>
        name (_arg1_, _arg2_, … _argN_)
        <statements>
```

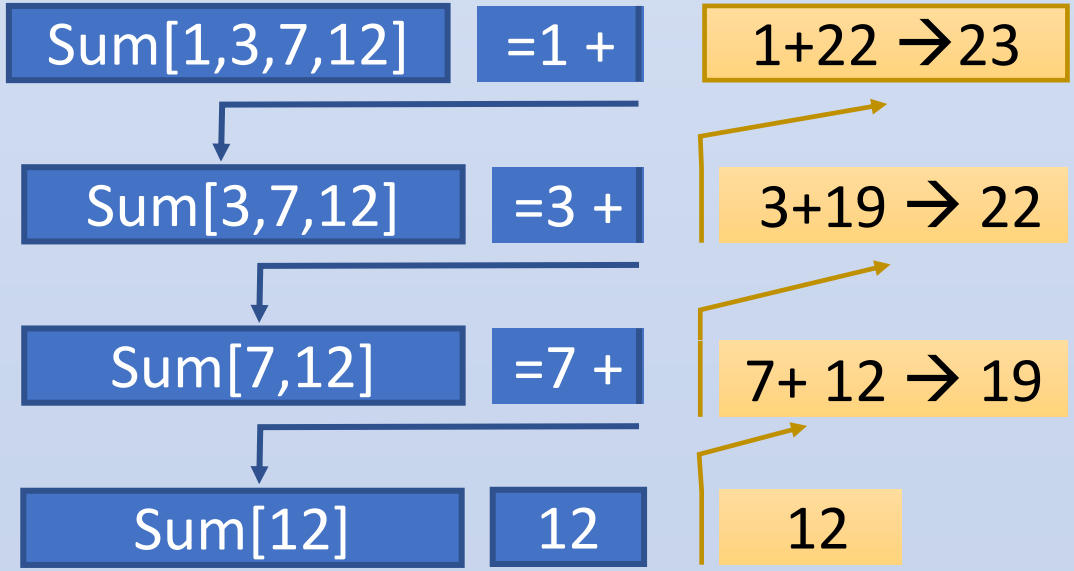
РЕКУРСИВНА ФУНКЦІЯ

```
def listsum (L) :
    if len(L)==1:
        return L[0]
    else:
        return L[0]+listsum(L[1 :])
```

Нерекурсивна гілка (вихід з рекурсії, база)

Рекурсивна гілка (тіло)

```
Print(listsum([1,3,7,12]))
```



РЕКУРСИВНА ФУНКЦІЯ

Глибина рекурсії - кількість рекурсивних входжень в функцію.
Обмеження - максимальна допустима глибина рекурсії.

Виключна ситуація
exception *RecursionError*

Керування максимальною глибиною рекурсії:

- **sys.*getrecursionlimit*()**
- **sys.*setrecursionlimit*(*limit*)**

НЕПРЯМИ ВИКЛИКИ ФУНКЦИІЇ

Python функція - **об'єкт**, можна працювати з ними, як зі звичайним об’єктом.

Об'єкти функції можуть: присвоюватися, передатися іншим функціям, зберігатися в структурах даних і так далі (подібно простим числам або рядками).

Об'єкти функції підтримують спеціальні операції: викликатися перерахуванням аргументів в (), наступних відразу ж за виразом функції.

Після виконання ***def()***, ім'я функції є посилання на об'єкт - її можна привласнити іншим іменам і викликати функцію за допомогою одного з них.

ІНТРОСПЕКЦІЯ

ІНТРОСПЕКЦІЯ – можливість для будь-якого об'єкта (функції також) отримати всю інформацію про його внутрішню структуру і середовищі виконання.

Дві групи: а) стандартні можливості (описані в документації по мові),
б) нестандартні (характерні для конкретної реалізації мови (наприклад, ***CPython***).

АНОТАЦІЇ

Анотації (короткий опис) **не мають ніякого семантичного значення**, використовуються в Python тільки для підтримки інформативності коду та його автоматизованого аналізу. Анотування це опція , не вимога мови.

```
def <name>(arg1 : expr,                    \
           arg2 : expr = value,            \
           ,…,*args : expr                 \
           *kwargs : expr ) : ->expr
    <statements>
```

Анотація аргументу

Анотація результату

PEP 3107 – Function annotation – синтаксис анотацій функцій.

АНОТАЦІЇ

Доступ до анотації через атрибут функції **__annotations__**

Вертає словник.

```
def foo(a: 'x', b: 5 + 6, c: list) -> max(2, 9): ...

foo.__annotation__
{'a': 'x', 'b': 11, 'c': list, 'return': 9}
```

PEP 3107 – Function annotation – синтаксис анотацій функцій.

АНОНІМНІ ФУНКЦІЇ (*lambda*)

Створення об'єкту «функція» в формі виразу.
Lambda – це вираз!
Lambda - складається з одного виразу!

Lambda - вираз вертає функцію, але **НЕ зв'язує** створений об'єкт (функцію) з іменем (змінною).

Головне: можна використовувати там, де def неможливо. Наприклад, в інших виразах.

LAMBDA ВИРАЗ

lambda_expr ::=

***lambda* arg1, arg2,…,argN : some_expression**

Lambda вираз

def lamda (arg1, arg2,…,argN) :

***return* some_expression**

Еквівалентна функція

Lambda - вираз не може містити анотацій та інші вирази.

Аргументи та області видимості аналогічні def…/

ОТОБРАЖЕННЯ НА ПОСЛІДОВНІСТЬ (map)

map_function ::=

map **(func, iterable1, iterable2, … iterableN)**

def func(arg1, arg2, … argN)

Застосовує **func** до кожного елементу ітеруємої послідовності/ послідовностей.

python ver < 3.0 -> list
python ver 3.0+ -> iterator

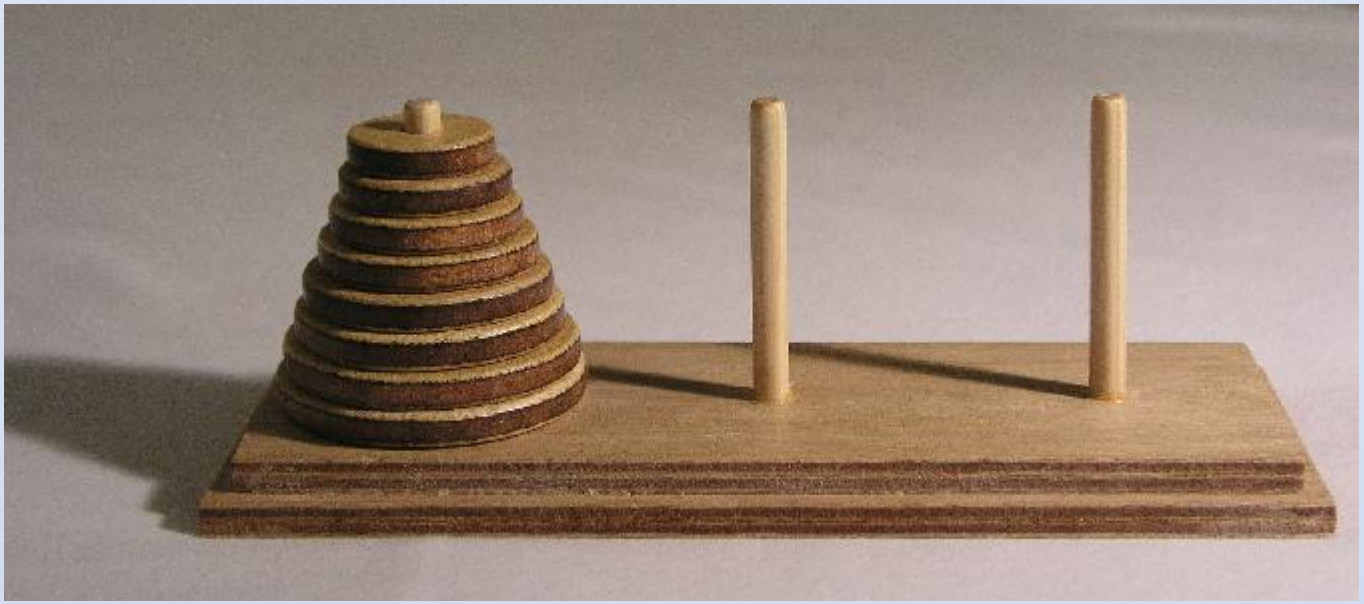
ОТОБРАЖЕННЯ НА ПОСЛІДОВНІСТЬ (map)

Не рекомендовано

! filter () ver < 3.0

! reduce () ver < 3.0

ХАНОЙСКІ БАШТИ



Дано три стрижня, на один з яких нанизані вісім кілець (n), причому кільця відрізняються розміром і лежать менше на більшій.
Завдання полягає в тому, щоб перенести піраміду з восьми кілець за найменше число ходів на інший стрижень. За один раз дозволяється переносити тільки одне кільце, причому не можна класти більше кільце на меншу.

ХАНОЙСКІ БАШТИ

Написати програму з використанням рекурсивного виклику функції.

https://ru.wikipedia.org/wiki/%D0%A5%D0%B0%D0%BD%D0%BE%D0%B9%D1%81%D0%BA%D0%B0%D1%8F_%D0%B1%D0%B0%D1%88%D0%BD%D1%8F

Контрольні питання

- Наведіть визначення рекурсивної функції в мові Python. Надайте приклади створення та використання рекурсивних функцій.

- Визначте поняття анотації функції, надайте приклади створення анотацій та їх використання.

- Наведіть визначення lambda виразу, надайте приклади створення та використання **lambda** виразів.

The END
Лекція 07

ЛЕКЦІЯ 08. PYTHON #7

1. **Інтерпретація скриптів Python. Байт-код.**
2. **Модулі Python.**
3. **Пакети Python.**

ІНТРПРЕТАЦІЯ СКРИПТІВ

Інтерпретатор (interpreter) – програма, необхідна для виконання інших програм, вид транслятору, який здійснює пооператорну (покомандну, построкову) обробку, **перетворення** у машинний код та **виконання** програми.

Компілятор (compiler) — програма (набір програм), що перетворює (компілює) деякий вхідний код, написаний певною мовою програмування (*source language*), на семантично еквівалентний код в іншій мові програмування (цільова мова, *target language*), з подальшою можливістю виконання програми комп'ютером.

ІНТРПРЕТАЦІЯ СКРИПТІВ

Простий інтерпретатор – аналізує і **відразу виконує** (власне інтерпретація) програму покомандно (або порядково), по мірі надходження тексту програми на вхід інтерпретатора.

Інтерпретатор компілюючого типу — це система з компілятора , який перекладає текст програми в деяке проміжне представлення (найчастіше в **байт-код**), і власне інтерпретатора, який виконує отриманий проміжний код - так звана віртуальна машина.

Загальна думка: машинний код набагато швидше, але байт-код краще захищений та є переносним.

ВИКОНАННЯ PYTHON ПРОГРАМИ

Вхідний текст

```
Def foo(x,y)
    X = 5
    Y = 7
    Z=X*Y
    PRINT(Z)
```

my_mod.py

Компіляція

Байт – код

b'd\x01}\x02d\x02}\x03|…..

```
0 LOAD CONST
2 STORE_FAST
4 LOAD CONST
6 STORE_FAST
...
 8 LOAD_FAST
10 LOAD_FAST
12 BINARY MULTIPLAY
...
20 CALL FUNCTION
```

my_mod.pyc

Виконання

PVM

Інтерпретація

35

РЕАЛІЗАЦІЇ PYTHON

Implementation	Virtual Machine	Compatible Lang
CPython	CPython VM	C
Jython	JVM	Juva
IronPython	CLR	C#
Brython	Javascript Eng.	JavaScript
RubyPython	RubyVM	Ruby
PyPy		Python

JupyterLab Spyder		
IPython / IP[y]	An enhanced Interactive Python. Last: 8.17.2 Released on Oct 31, 2023.	C

https://ipython.readthedocs.io/en/stable/index.html

МОДУЛІ

Типова програма ділиться на кілька файлів для полегшення обслуговування. Також можете використати зручну функцію, яка написана раніше в кількох програмах, не копіюючи її визначення в кожну програму.

Для підтримки цього Python має спосіб помістити визначення у файл і використовувати їх у скрипт або в інтерактивний екземпляр інтерпретатора. Такий файл називається модулем. Визначення з модуля можна імпортувати в інші модулі або в головний модуль (набір змінних, до яких ви маєте доступ у скрипті, що виконується на верхньому рівні та в режимі калькулятора).

МОДУЛІ

Модуль — це файл, що містить визначення та оператори Python. Ім'я файлу — це ім'я модуля з суфіксом .py. Усередині модуля ім'я модуля (у вигляді рядка) доступне як значення глобальної змінної __*name*__.

https://docs.python.org/3/tutorial/modules.html

МОДУЛІ

modul_1.py

```
def func(mssg):
    print('Mod 1', mssg)
```

modul_2.py

```
def func(mssg):
    print('Mod 2', mssg)
```

my_main.py

```
import modul_1
modul_1.func('OhOhOh')
```

МОДУЛІ

my_main.py

```
import modul_1
import modul_2
modul_1.func('OhOhOh')
modul_1.func('HaHaHa')
```

modul_1.py

```
def func(mssg):
    print('Mod 1', mssg)
```

modul_2.py

```
def func(mssg):
    print('Mod 2', mssg)
```

ПРОСТІР ІМЕН МОДУЛЯ

МОДУЛЬ → ПАКЕТ ІМЕН

- **Інструкції модуля виконуються** під час **першої** спроби імпорту. Перший імпорт - інтерпретатор Python створює **порожній** об'єкт модуля і виконує інструкції в модулі одну за одною, від початку файлу до кінця.
- **Операції присвоювання** (не вкладені в def () або class()), що виконуються на верхньому рівні, створюють атрибути об'єкта модуля і зберігаються в просторі імен модуля.
- Область видимості модуля (простір імен) після завантаження модуля перетворюється в **атрибут-словник об'єкта модуля**. Доступ до простору імен модуля можна отримати через атрибут **__dict__** або **dir (M)**.

ІНСТРУКЦІЇ ІМПОРТУ

	Дія
import <*module*>	*Файл* ***module*** *завантажується та перетворюється в* ***ім'я змінної****, яка посилається на повний* ***об'єкт*** *модуля із усіма його атрибутами*
import <*module*> as <*mod_name*>	***ім'я змінної := mod_name*** *(нові, короткі імена)*
from <*module*> import *	*Завантажуються* ***всі*** *об'єкти-функції модуля, перетворюються в імена змінних та копіюються в область видимості програми*
4. from <*module*> import <*fun1, fun2, …*>	*…. Імпортуються окремі функції*
5 from <*module*> Import <*fun1, fun2,…*> as F1,F2,..	*…. Імпортуються окремі функції з новими іменами*

ІНСТРУКЦІЇ ІМПОРТУ

import , from → !!! інструкції (як і всі інші)

from ..!.. вставляє простір імен модуля в простір імен програми.

import ..!.. створює **єдине** ім'я в просторі імен програми що посилається на об’єкт-модуль (на простір імен модуля).

! ПЕРЕВАГУ інструкції **import ….**

Функція	Дія
from *imp* import *reload*	**reload** примусово виконує повторну завантаження вже завантаженого модуля і запускає його програмний код.
***reload* <module>**	Інструкції присвоювання, що виконуються при повторному запуску, будуть змінювати існуючий об'єкт модуля.

ІМПОРТ МОДУЛЯ

Крок	Дія
1.Пошук	*Пошук модуля в робочому середовищі*
2.Компіляція	*Перетворення в байт-код (*.pyc), якщо це необхідно*
3.Запуск	*Виконується байт-код, створюються всі об'єкти модуля та, відповідно, простір імен модуля (атрибути всіх об'єктів модуля).*

ПОШУК МОДУЛЯ

Де шукати модуль ?

	Напрямки пошуку
1.	*Домашня директорія програми*
2.	*Змінна оточення **PYTHONPATH,** якщо є*
3.	*Каталоги стандартних бібліотек*
4.	*Зміст будь-яких файлів *.pht, якщо вони є*

import sys

sys.path – перегляд шляхів пошуку

СИНТАКСИС КВАЛІФІКАЦІЇ ІМЕН

Для доступу до атрибутів будь-якого об'єкта використовується синтаксис кваліфікації імені ***object.attribute*** **->** це вираз, що повертає значення, яке присвоєно імені атрибуту.

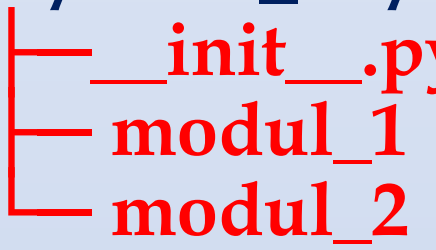
Звернення до імен, **кваліфікуючи їх**, явно вказує інтерпретатору об'єкт, атрибут якого потрібно використати.

- Прості змінні (правило **LEGB**), наприклад **X** - пошук імені **X** в поточних областях видимості.
- Кваліфіковані імена **X.Y** - пошук імені **X** спочатку в поточних областях видимості, а потім буде виконаний пошук атрибута **Y** в об'єкті **X** (не в областях видимості).
- Кваліфіковані шляхи **X.Y.Z** - буде проведений пошук імені **Y** в об'єкті **X**, а потім пошук імені **Z** в об'єкті **X.Y**.

ПАКЕТ МОДУЛІВ

ПАКЕТ → директорія (каталог), в якому розташовані модулі

.../work_dir/dir



import dir.modul_1

__init__.py призначений для виконання дій по **першій** ініціалізації пакету, створення простору імен для каталогу і реалізації поведінки інструкцій імпорту, наприклад створення файлів, підключення до БД та інше.

МОДУЛЬ як СКРИПТ

Кожен модуль має вбудований атрибут **__name__**, який встановлюється

Або → <*modul*>, якщо файл модуля імпортується

Або → "__main__" , якщо файл модуля запускається як головний файл програми.

Тобто модуль може перевірити власний атрибут **__name__** і визначити, чи був він запущений як самостійна програма або імпортовано іншим модулем.

В модулі можна

```
if __name__ == "__main__":
        ….. # що потрібно
        …..
```

Контрольні питання

- Надайте визначення компілятора та інтерпретатора. Наведіть порядок виконання Python скриптів.
- Обґрунтуйте необхідність та вкажіть переваги використання модулів.
- Надайте перелік інструкцій імпорту модулів, поясніть їх відмінності та наведіть відповідні приклади.
- Наведіть способи завдання шляхів пошуку модуля.
- Поясніть сутність поняття кваліфіковані імена та наведіть відповідні приклади.
- Надайте призначення пакетів модулів та поясніть порядок створення пакетів.

The END
Лекція 08

ЛЕКЦІЯ 09. PYTHON #8

1. ООП в Python.
2. Створення класу.
3. Простір імен.
4. Дерево класів, успадкування.
5. Вбудовані аргументи.

КЛАСИ

Класи Python реалізують усі основні концепції ООП:

- **Інкапсуляція**: об'єднання даних та методів
- **Спадкування:** створення нового класу на базі існуючого
- **Поліморфізм:** використання об'єктів з однаковим інтерфейсом без інформації про тип і внутрішню структуру об'єкта

КЛАСИ

КЛАС Python

→ об'єкт Python

ЕКЗЕМПЛЯР (інстанс, instance) КЛАСУ

→ об'єкт Python

!!! Різні об'єкти !!!

<table>
<tr><th>C++</th><th colspan="2">Python</th></tr>
<tr><td>Поле</td><td rowspan="3">АТРИБУТ</td><td>Змінна (вказівник)</td></tr>
<tr><td rowspan="2">Метод
(процедура,
функція)</td><td rowspan="2">Функція,
визначена в класі</td></tr>
<tr></tr>
</table>

СТВОРЕННЯ КЛАСУ

class <name> (superclass, …):
<тіло класу>

Кожна інструкція **class** створює новий об'єкт типу **клас.**

Кожний раз, коли викликається клас, він створює новий об'єкт – **екземпляр** класу.

Екземпляри автоматично зв'язуються с класом із яких вони були створені.

Класи зв'язані із своїми суперкласами, які вказані в круглих дужках в інструкції **class.** Послідовність суперкласів у списку визначає порядок розташування в дереві пошуку імен.

СТВОРЕННЯ КЛАСУ

class <name> (superclass, …):
<тіло класу>

class – і**нструкція** створення об'єкту, посилання на який зберігається в просторі імен як **<ім'я класу>.**

<тіло_класу> сукупність інструкцій. Може містити присвоювання змінним **data=value** (**data** стає **атрибутом - полем** класу) та визначення функцій **def …(self, …): ….** (стає **атрибутом - методом** класу).

Перший аргументом методу завжди є екземпляр класу, для якого викликається цей метод. За угодою, цей аргумент називається **"self"**. Спеціальний метод **__init__( )** – конструктор – викликається автоматично при створенні екземпляра класу.

СТВОРЕННЯ КЛАСУ

```
class MyClass :
    def __init__(self, val)
        self.value = val

    def prntvl (self)
        print('value =', self.value)

Myobj = MyClass(3.14)
Myobj.prntvl()
Myobj.value = "HOHOHO"
Myobj.prntvl()
```

СТВОРЮЄ об'єкт типу **КЛАС**

Конструктор

Функція класу

СТВОРЮЄ об'єкт типу **ЕКЗЕМПЛЯР КЛАСУ**

Звернення до аргументу (методу)

→ *3.14*

Звернення до аргументу (поле)

→ *HOHOHO*

ПРОСТІР ІМЕН

Виклик об'єкта класу як функції створює новий об'єкт екземпляру.

Кожен об'єкт екземпляру успадковує атрибути класу і набуває **свій власний простір імен.** Вони спочатку **порожні**, але успадковують атрибути класів, з яких були створені.

Методи класу отримують в першому аргументі (з ім'ям ***self***) посилання на оброблюваний об'єкт – екземпляр. Присвоювання атрибутів через посилання ***self*** створює чи змінює дані **екземпляру**, а не класу.

ПРОСТІР ІМЕН

```
class Cls_Empty :
    pass
```

Список імен поточної області видимості

```
dir()
```
→ *['In', 'Out' , ..., Cls_Empty',...]*

```
dir(Cls_Empty)
```

Список атрибутів класу

→ *['__class__', '__doc__' , '__dict__',.....]*

```
Cls_empty.__dict__
```

Словник класу

→ *({'__module__': '__main__' ,*
'__doc__' : None',
__dict__': <...>, ... })

```
Cls_empty.X = 25; Cls_empty.Y = 42
Cls_empty.__dict__
```

→ *({..., 'X' : 25, 'Y': 42})*

Словник класу
!!! X Y

ПРОСТІР ІМЕН

```
ex_1 = Cls_empty()
ex_2 = Cls_empty()
```

2 екземпляри класу

```
ex_1.X        → 25
ex_2.Y        → 42

ex_1.X = 'HOHOHO'
ex_2.Y = '146823'

ex_1.X        → 'HOHOHO'
ex_1.Y        → 42

ex_2.X        → 25
ex_2.Y        → 146823
```

Змінюються атрибути екземплярів, но не класу

```
cl_empty.X → 25

cl_empty.Y → 42
```

ПРОСТІР ІМЕН

```
class MixedData :
    data = 'STRING'
    def __init__(self, value):
        self.data = value

def dispout(self):
        print (self.data, MixedData.data)

x = MixedData ('FIRST')
y = MixedData (2)

x.dispout()
```

→ *FIRST STRING*

```
y.dispout()
```

→ *2 STRING*

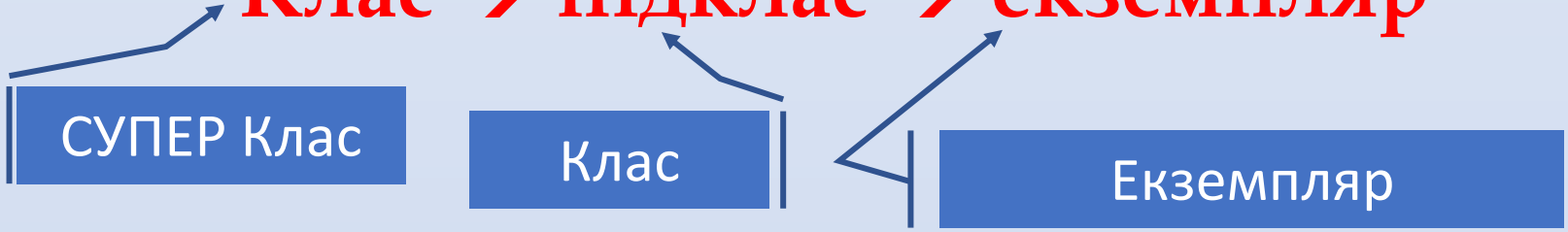
data - атрибут класу

data - атрибут екземпляру

Функція друку атрибуту класу та атрибуту екземпляру

СПАДКУВАННЯ: СУПЕРКЛАС, ПІДКЛАС

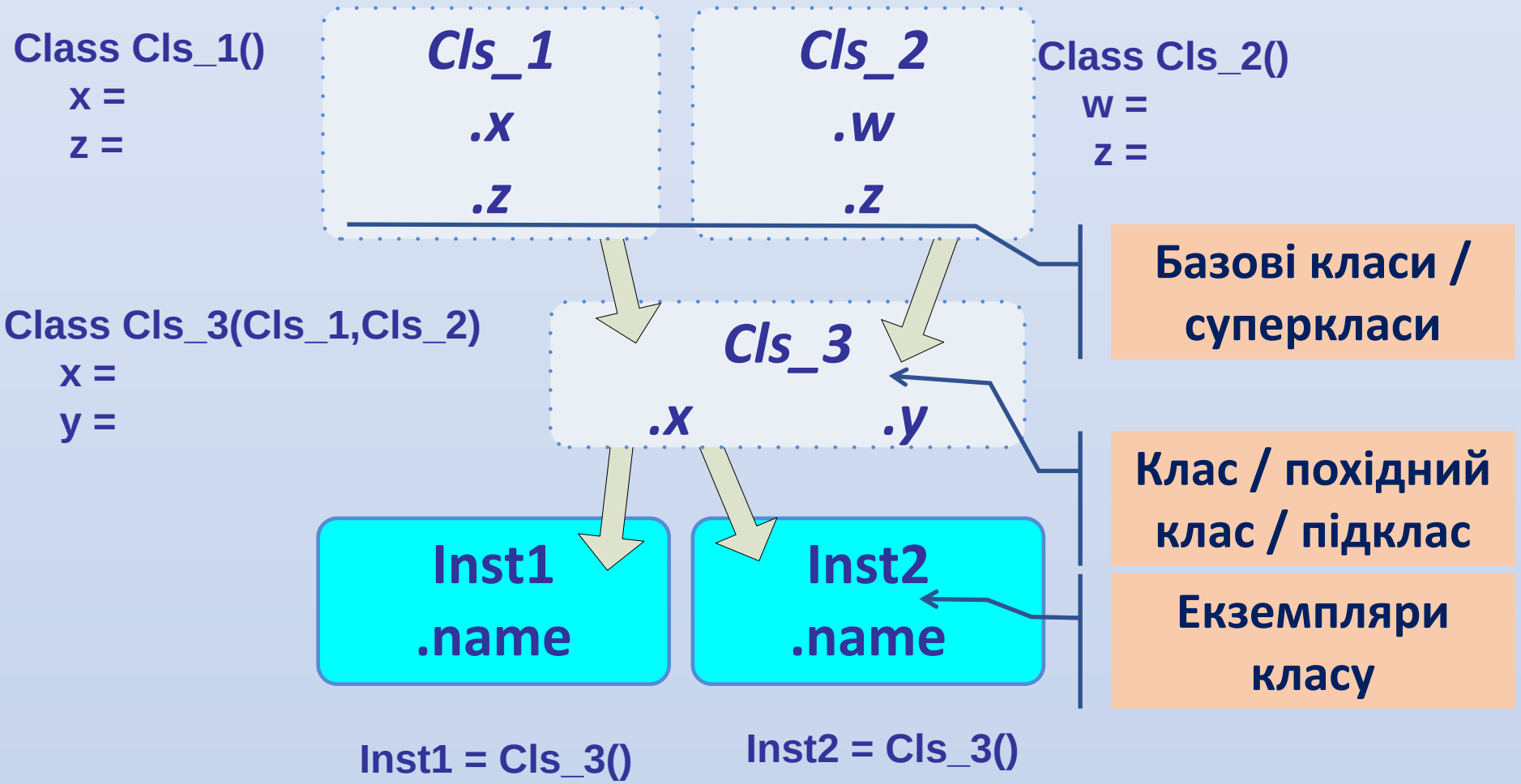
Клас → підклас → екземпляр



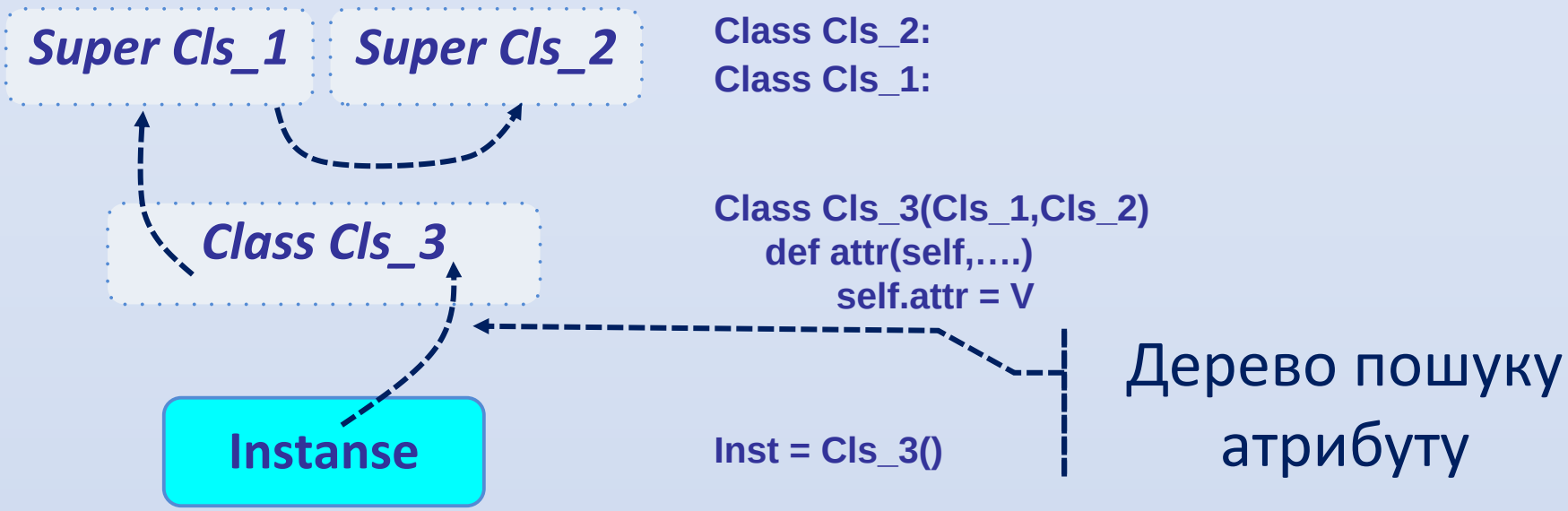
Суперкласи перераховуються в круглих дужках в заголовку інструкції ***class***. Клас, що наслідує, називається **підкласом**, а клас, який унаслідується, називається його **супер-класом**.

Клас наслідує **атрибути суперкласу, екземпляр** наслідує **атрибути класу.**

ДЕРЕВО КЛАСІВ: СУПЕРКЛАС, КЛАС, ЕКЗЕМПЛЯР



ДЕРЕВО УСПАДКУВАННЯ



Атрибути екземплярів створюються за допомогою присвоювання атрибутів аргументу ***self*** в методах класу. Атрибути класів створюються інструкціями (привласнення), розташованої всередині інструкції **class**.

Посилання на суперклас - перерахування класів в круглих дужках в заголовку інструкції **class**.

СИНТАКСИС КВАЛІФІКАЦІЇ ІМЕН

Для доступу до атрибутів будь-якого об'єкта використовується синтаксис кваліфікації імені ***object.attribute*** **->** це вираз, що повертає значення, яке присвоєно імені атрибуту.

Звернення до імен, **кваліфікуючи їх**, явно вказує інтерпретатору об'єкт, атрибут якого потрібно використати.

- Прості змінні (правило **LEGB**), наприклад **X** - пошук імені **X** в поточних областях видимості.
- Кваліфіковані імена **X.Y** - пошук імені **X** спочатку в поточних областях видимості, а потім буде виконаний пошук атрибута **Y** в об'єкті **X** (не в областях видимості).
- Кваліфіковані шляхи **X.Y.Z** - буде проведений пошук імені **Y** в об'єкті **X**, а потім пошук імені **Z** в об'єкті **X.Y**.

СИНТАКСИС КВАЛІФІКАЦІЇ ІМЕН
додаток для класів

Пошук в дереві успадкування кваліфікованих імен типу ***object.X***

Посилання *object.X* → пошук атрибута **X** виконується спочатку в об'єкті ***object***, потім у всіх класах, розташованих вище в дереві спадкоємства. 194

Присвоєння *object.X* = *value* → створюється чи змінюється атрибут з ім'ям **X** в просторі імен об'єкта ***object*, і нічого більше**.

ВБУДОВАНІ АРГУМЕНТИ КЛАСУ

Атрибут	
__name__	Ім'я класу
__module__	Ім’я модуля
__dict__	Словник атрибутів класу
_bases__	Кортеж суперкласів
__doc__	Рядок документації класу

ВБУДОВАНІ АРГУМЕНТИ ЕКЗЕМПЛЯРУ

Атрибут	
__dict__	**Словник атрибутів класу**
__class__	**Клас, що є «батьком» екземпляру**
__init__	**Конструктор**
__del__	Деструктор (фіналізатор)
__cmp__	Викликається для порівняння (нема в версіях >=3.0)
__hash__	Хеш-значення обєкту
__getattr__	Вертає значення атрибуту
__setattr__	Встановлює значення атрибуту
__delattr__	Видаляє атрибут
__call__	Виконується при виклику екземпляру

ВБУДОВАНІ АРГУМЕНТИ (приклад: __class__, __bases__)

```
def clstree (cls, indent) :
    print (‘.’ * indent  + cls.__name__)
    for supercls  in cls.__bases__ :     #!рекурсія суперкласів
        clstree(supercls, indent+4)
def instancetree (inst) :
    print (‘tree of ’, inst)
    clastre (inst.__class__, 4)

class A              : pass
class B(A)           : pass
class C(A)           : pass
class D(B,C)         : pass
class E              : pass
class F(D,E)         : pass

instancetree (B())                  # що буде надруковано?
```

Контрольні питання

- Обґрунтуйте необхідність та вкажіть переваги використання класів.
- Поясніть відмінності класу і екземпляру класу. Надайте приклад створення класу и екземпляру.
- Поясніть призначення об'єкту **self** та надайте приклади його застосування.
- Поясніть використання аргументу **__init__** при створенні класу, надайте приклади.
- Поясніть призначення вбудованого аргументу класу **__bases__** .
- Поясніть призначення вбудованого аргументу класу **__doc__** .
- Поясніть призначення вбудованого аргументу екземпляру **__dict__** .

The END
Лекція 09

ЛЕКЦІЯ 10. PYTHON # 9

1. Перевантаження операторів
2. Об'єкт, що ітерується
3. Ітератор
4. Генератор

ПЕРЕВАНТАЖЕННЯ

Перевантаження операторів в Python:

- дозволяє класам брати участь в звичайних операціях,
- класи можуть перевантажувати всі оператори виразів,
- класи можуть також перевантажувати такі операції, як вивід, виклик функцій, звернення до атрибутів і …
- перевантаження полягає в реалізації в класах методів із спеціальними іменами.

!!! якщо в класі визначено метод з спеціальним ім'ям, інтерпретатор буде автоматично викликати його при виконання відповідного методу екземпляру класу .

ПЕРЕВАНТАЖЕННЯ

Імена методів, що починаються і закінчуються двома символами підкреслення (__**X**__), мають спеціальне призначення.

Мова Python визначає фіксовані і незмінні імена методів **для кожної** з операцій. Перевантаження операторів реалізовано за рахунок створення методів зі цими спеціальними іменами для перехоплювання операцій.

Такі методи викликаються автоматично, коли екземпляр бере участь у вбудованих операціях.

МЕТОДИ ПЕРВАНТАЖЕННЯ

Метод	Перевантажує	Викликається
__init__	Конструктор	При створенні об'єкту
__del__	Деструктор	При знищенні об'єкту
__repr__ **__str__**	Вивід, перетворення (строкове представлення)	print(), repr(), str(), format ()
__len__	Довжина	len(x)
__bool__	Перевірка логічного значення	
__lt__, __gt__, **__le__, __ge__,** **__eq__, __ne__,**	Порівняння	X < Y, X <= Y X > Y, X >= Y X == Y, X != Y

ВБУДОВАНІ АРГУМЕНТИ (__init__)

Метод ***__init__*** є **конструктором,** який використовується для ініціалізації стану об'єкту.

Метод ***__init__*** завжди викликається при створенні нового екземпляру класу. Якщо метод ***__init__*** відсутнє в класі, інтерпретатор виконує пошук в суперкласі.

Важливо:
В методі ***__init__*** класу можливо викликати ***__init__*** ***суперкласу*** , за допомогою кваліфікованого ім'я.

Див: EXAMPLES LEC 10 Class 1

ВБУДОВАНІ АРГУМЕНТИ (__str__ , __repr__)

Строкове представлення об'єктів за замовчуванням має не містить корисної інформації і нелегко сприймається. Методи ***__repr__ , __str__*** дозволяють визначити більш легкий для читання формат виведення екземплярів об'єктів.
Методи повертають строкове представлення екземпляру, завжди використовується для перетворення об'єкта ***self.data*** в рядок
Метод **__repr__** використовується всюди, **за винятком функцій *print()* і *str()***, якщо визначено метод **__str__**. Якщо метод ***__str__*** відсутен, операції виведення використовуватимуть метод ***__repr__***, але не навпаки.

Див: EXAMPLES LEC 10 Class 2

АРИФМЕТИКО-ЛОГІЧНІ

Метод	Перевантажує	Викликається
__add__, __radd__, __iadd__	Оператор +	Додавання
__sub__	Оператор -	Віднімання
__mul__	Оператор *	Множення
__truediv__	Оператор /	Ділення
__flordiv__	Оператор //	Цілочислове ділення
__mod__	Оператор %	Залишок ділення
__divmod__	Оператор divmod(X,Y)	Частка ділення
__pow__	Оператор **	степінь
__and__	Оператор &	«ТА»
__or__	Оператор ^	«АБО»
__xor__	Оператор |	«Виняткове АБО»

ІТЕРАТИВНІСТЬ. ВИЗНАЧЕННЯ

Ітеруємий обєкт (об'єкт, що ітерується, iterable, iterable object) - об'єкт, який має метод **__iter __ ( )**, який повертає відповідний об’єкт - ітератор.

Ітератор (iterator) - об'єкт, який повертається методом **__iter __ ()** і має метод **__next __ ()**, що витягує (видає) наступний елемент контейнеру. При цьому цей елемент уже не міститься в ітераторі - ітератор в кінцевому підсумку спустошується і вертає помилку **StopIteration**.

ІТЕРАТИВНІСТЬ. ВИЗНАЧЕННЯ

Протокол ітерацій - об'єкт реалізує метод **_next _**, який вертає наступне значення послідовності і генерує виключну ситуацію *StopIteration*

ІТЕРАТИВНІСТЬ

Перевірка: є об'єкт ітеруємим ?

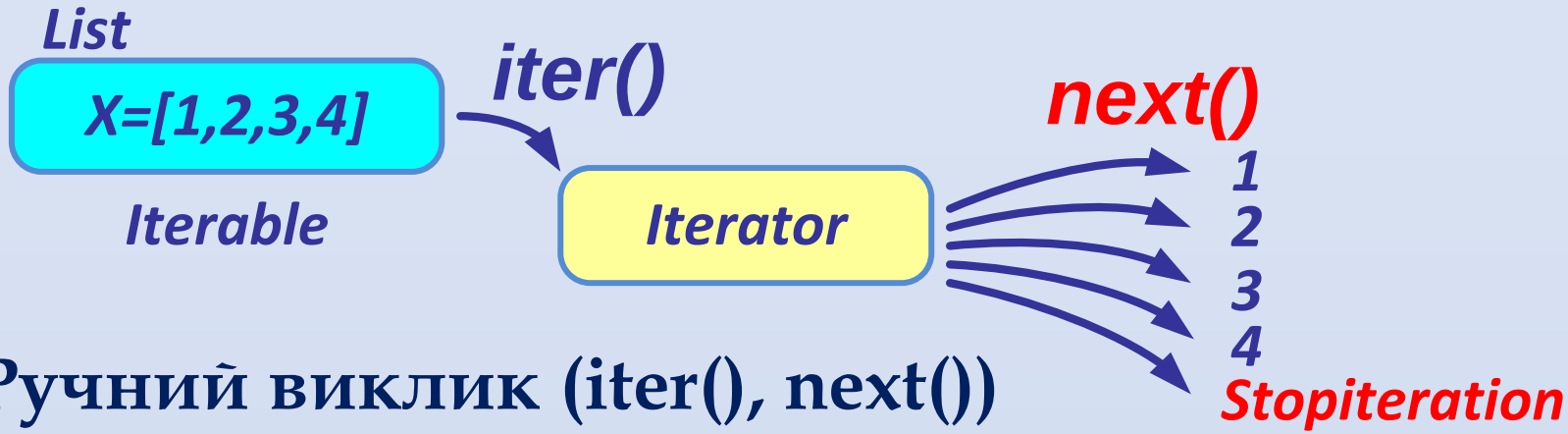
hasattr(object , ‘__iter __’)

Приклад :

hasattr(str , ‘__iter __’) →*True*

hasattr(bool , ‘__iter __’) →*False*

ITEPATOP

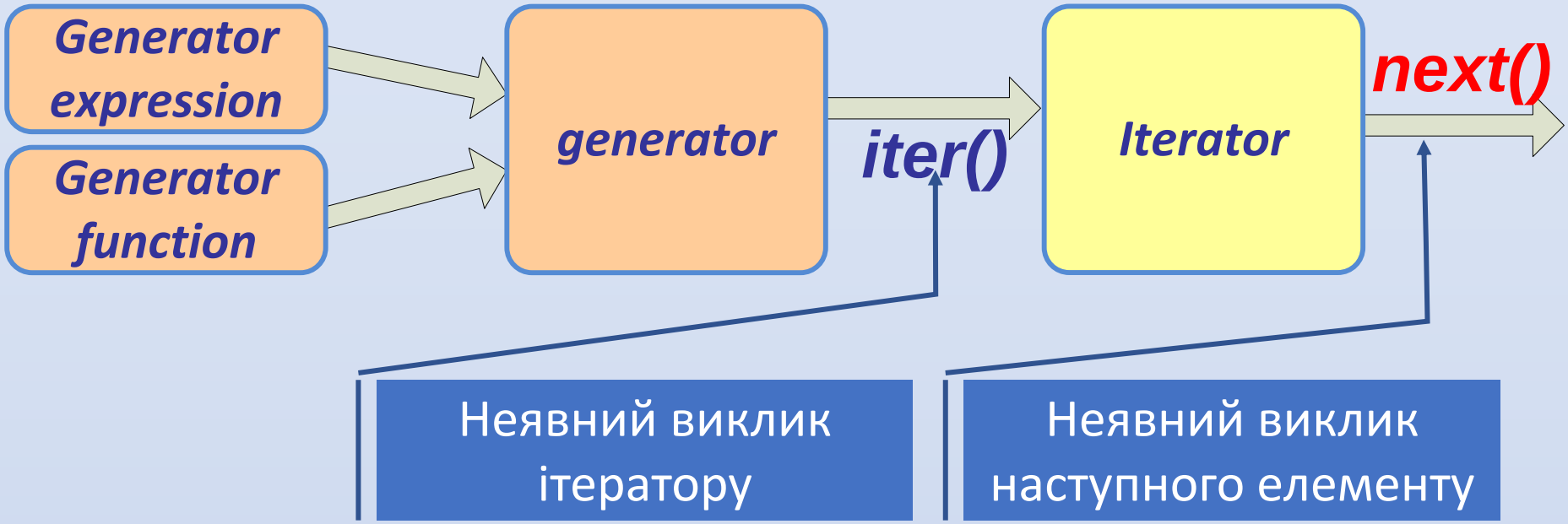


- **Ручний виклик (iter(), next())**
- **Ручний виклик WHILE**
- **Автоматичний виклик FOR**

for *item* in *X* : print *(item)*

→*1*
→*2*
→*3*
→*4*

ГЕНЕРАТОР



Генератор створює методи __iter__ , __next__ автоматично!

Два типи генераторів:

1. **generator expression** - генератор - вираз
2. **generator function** - генератор - функція

ІТЕРАТИВНІСТЬ. ВИЗНАЧЕННЯ

Генератор (**generator, generator expression**) - спеціальний клас **функцій**, який дозволяє легко створювати свої ітератори.
На відміну від звичайних функцій, генератор не просто повертає значення і завершує роботу, а повертає **ітератор**, який віддає елементи по одному.
Генератор витягує (видає) значення. При цьому значення повертаються за запитом, і після повернення одного значення виконання функції - генератора **припиняється** до запиту наступного значення.
Між запитами генератор зберігає свій стан.

ГЕНЕРАТОР ФУНКЦІЯ

Головною особливістю **генератора** є повернення елементів на вимогу.

Генератор – функція забезпечує зручний спосіб реалізації протоколу ітерацій. **Генератор** - це ітерабельний об'єкт, створений за допомогою функції із **yield** інструкцією виходу.

Відмінність:
Звичайна функція припиняє своє виконання, коли вона виконує інструкцію виходу **return.**

Функція з інструкцією виходу **yield** зберігає свій стан, який може бути «підхоплений» наступного разу, коли викликається.

ГЕНЕРАТОР ВИРАЗ

Генератор–вираз – спрощений засіб створення генератору (порівняно з генератором – функцією).

Генератор – вираз дозволяє створювати генератор «в польоті» без використання інструкції виходу **yield.**

Генератор – вираз може бути записано з використанням синтаксису, схожого на синтаксис компоновки словника, але не в квадратних, а в круглих дужках.

ІНСТРУКЦІЯ FOR … IN …

for *target_list* **in** *expression_list* **:** *suite*
[else : *suite***]**

Suite – група (блок) інструкцій

expression_list - об'єкт, що ітерується. Ітератор створюється після обчислення expression_list. *Suite* виконується послідовно для кожного елемента що вертає ітератор. Коли елементи послідовності вичерпано (ітератор вертає *StopIteration,* виконується *suite* в виразі **[else :** *suite***]** (якщо присутнє) і виконання циклу завершується.

https://docs.python.org/3/reference/compound_stmts.html#the-for-statement

Контрольні питання

- Наведіть переваги використання перевантаження операторів.
- Поясніть призначення методів **__str__**, **__repr__** та надайте приклади їх застосування.
- Поясніть призначення методу **__add__** та надайте приклади його перевантаження.
- Надайте визначення об'єкта, що ітерується, ітератора, генератора.
- Поясніть протокол ітерації в мові Python. Поясніть призначення методів **__iter __ ,__next __.**
- Наведіть приклад створення класу, що ітерується.
- Надайте визначення генератора – функції та генератора – виразу, поясніть їх відмінності. Наведіть приклади.

The END
Лекція 10

ЛЕКЦІЯ 11. PYTHON # 10

1. **Виключні ситуації (Try … except)**
2. **PEP** https://www.python.org/dev/peps/

ВИНЯТКОВІ СИТУАЦІЇ

Виняток (виняткова ситуація, exception) – спеціальний даних в Python.
Винятки повідомляють про помилки програмі.

Програмна обробка необхідна щоб програма не завершувалося аварійно кожен раз, коли виникає виняток. Для цього блок коду, в якому можлива поява виняткової ситуації необхідно помістити всередину спеціальної синтаксичної конструкції

try … except …
try … except … else ….
try … except … else … finally …

https://docs.python.org/3/library/exceptions.html

ВИНЯТКОВІ СИТУАЦІЇ

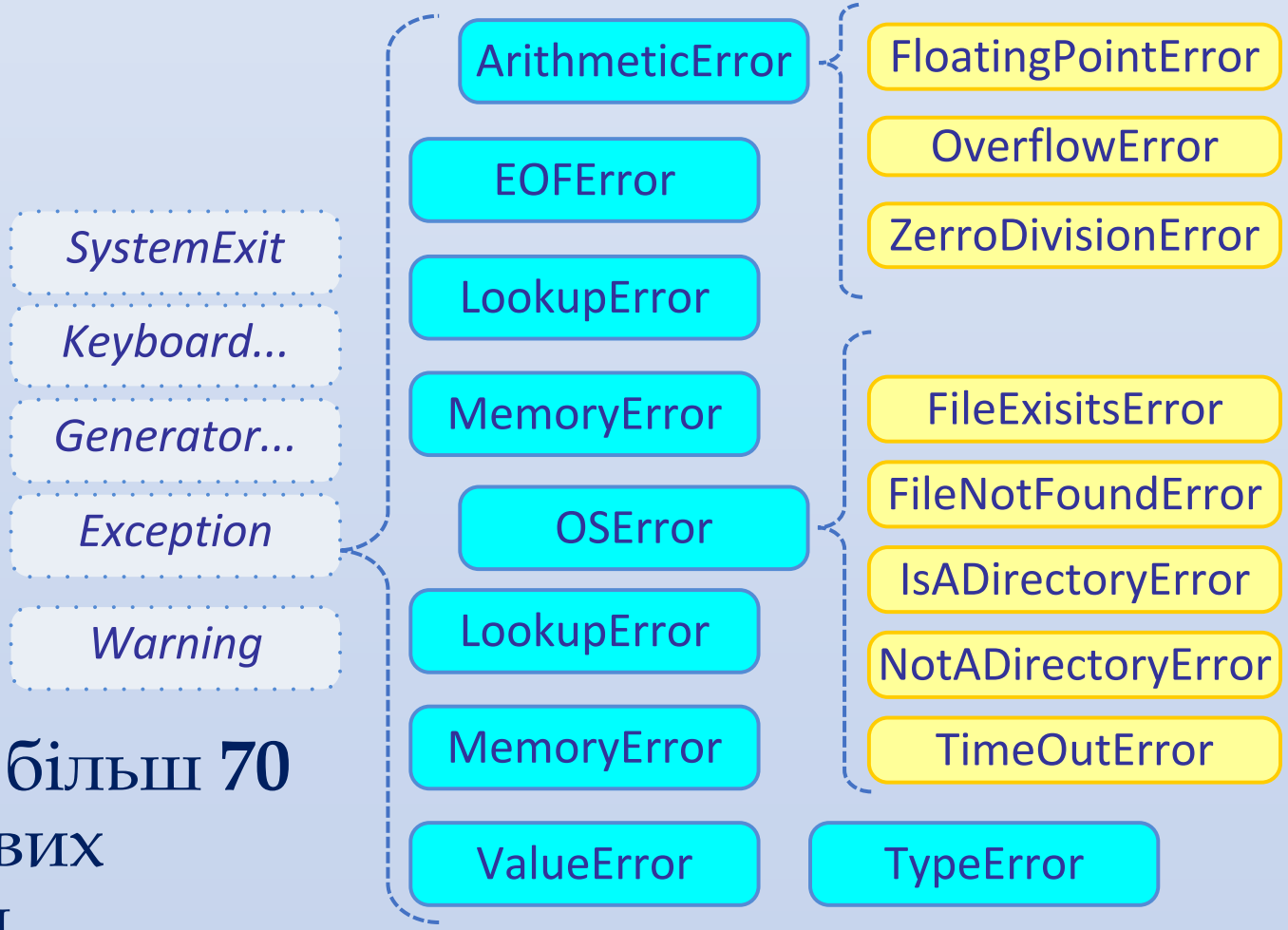
```
try:
   block-1
except Exception1:
   handler-1
except  Exception2:
   handler-2
else:
   else-block
finally:
   final-block
```

Виконується **block-1**. Якщо створюється виняткова ситуація перевіряється `except` блок.
Якщо виняток - **Exception1** – виконується блок **handler-1**.
Якщо виняток - **Exception2** – виконується блок **handler-2**
і так далі.

Якщо виняток не створюється, виконується **else-block** (якщо присутній). В будь якому випадку один раз виконується **final-block** (якщо присутній).

ВИНЯТКОВІ СИТУАЦІЇ

Exception1, Exception2 – ім'я винятковової ситуації (спеціальний тип даних **Python**)



Загалом більш **70** винятковых ситуацій

ГЕНЕРУВАННЯ ВИНЯТКІВ

Інструкція `raise` дозволяє примусово генерувати виняткової ситуації

`raise` Exception — **Exception –** вказує на тип виняткової ситуації.

Python дозволяє створювати власні виняткові ситуації Для цього потрібно створити клас, який є підкласом класу **Exception**().
Важливо: в класі **Exception** визначено метод **__str__**, що дозволяє виводити значення його атрибутів.

здіймати, підняти

Приклади див: EXAMPLES LEC 12 PYTHON 11 Except

PEP

Python Enhancement Proposal (PEP) - механізмом документування проектних рішень, які пройшли в Python, і для пропозиції нових можливостей мови.

Meta-PEPs (PEPs about PEPs or Processes)

	PEP	PEP Title	PEP Author(s)
P	1	PEP Purpose and Guidelines	Warsaw, Hylton, Goodger, Coghlan
P	4	Deprecation of Standard Modules	Cannon, von Löwis

Other Informational PEPs

	PEP	PEP Title	PEP Author(s)
I	13	Python Language Governance	and community
I	20	The Zen of Python	Peters

> 8000

I	8101	2020 Term steering council election	Jodlowska, III

Посилання https://www.python.org/dev/peps/

PEP 8 – Стиль коду Python

PEP 8 -- Style Guide for Python Code

Ключова ідея: код **читається** набагато більше разів, ніж пишеться. Рекомендації про стиль написання коду спрямовані на те, щоб поліпшити читабельність коду і зробити його узгодженим між великим числом проектів. В ідеалі, весь код повинен бути написано в єдиному стилі, і будь-хто може легко його прочитати.

!!! PEP 20

«Читабельність має значення».

Посилання https://www.python.org/dev/peps/pep-0008/

PEP 8 – Стиль коду Python

Відступи:
4 **пробіла** на один рівень відступу. Ніколи не змішувати символи табуляції і пробіли. Рекомендовано тільки **пробіли.**

Максимальна довжина рядка.
Рекомендовано обмежити **79** символами. Переважний спосіб перенесення довгих рядків - **зворотний слеш.**
Рекомендовано ставити перенесення рядка після бінарного оператора, але не перед ним (математично перед бінарним оператором).

PEP 8 – Стиль коду Python

Пусті рядки:
Відокремлювати функції (верхнього рівня, що не функції всередині функцій) і визначення класів **двома** порожніми рядками.
Визначення методів всередині класу відокремлювати **одним** порожнім рядком.
Допускається використання порожніх рядків в коді функцій, щоб відокремити один від одного логічні частини.

PEP 8 – Коментарі

Блок коментарів:

Блок коментарів зазвичай пояснює код (весь, або тільки деяку частину), що йде після блоку, і повинен мати той же відступ, що і сам код. Кожен рядок такого блоку повинен починатися з символу # і одного пробілу після нього.

Абзаци всередині блоку коментарів краще відокремлювати рядком, що складається з одного символу #.

PEP 8 – Коментарі

Коментарі в рядку коду:
Коментарі в рядку з кодом не потрібні і лише відволікають від читання, якщо вони пояснюють очевидне. Намагайтеся рідше використовувати подібні коментарі.

Він повинен відділятися хоча б двома пробілами від інструкції. Він повинні починатися з символу # і пробілу.

PEP 8 – Документування

Рядки документації:
Повна угода про написання документації (docstrings) викладена в PEP 257.

Пишіть документацію для всіх модулів, функцій, класів, методів, які оголошені як public. Коментар потрібно писати після рядка з **`def name()`**:

```
""" вертає список імен
Додатковий аргумент – рік народження

"""
```

PEP 8 – Стилі імен Python

Визначення стилю:

- `b` - одиночна маленька буква;
- `B` - одиночна заголовна буква;
- `lowercase` - слово в нижньому регістрі;
- `lower_case_with_underscores` - слова з маленьких букв з підкресленнями;
- `UPPERCASE` - великі літери;
- `UPPERCASE_WITH_UNDERSCORES` - слова з великої літери з підкресленнями;
- `CapitalizedWords` - слова з великими літерами, або `CapWords`, або `CamelCase 5`. Іноді називається `StudlyCaps`. HTTPServerError краще, ніж HttpServerError.

PEP 8 – Стилі імен Python

Визначення стилю:

- **`mixedCase`** - відрізняється від **`CapitalizedWords`** тим, що перше слово починається з маленької літери; **`Capitalized_Words_With_Underscores`** - слова з великими літерами і підкресленнями
- **`__double_leading_and_trailing_underscore__`** - подвійне підкреслення на початку і в кінці імені. Наприклад, __init__, __import__ або __file__. Використовувати тільки так, як написано в документації.

PEP 8 – Стилі імен Python

Загальні рекомендації:

Ніколи не використовуйте символи `l` (малелькая латинська буква «ель»), `O` (заголовна латинська буква «o») або `I` (заголовна латинська буква «ай») як однобуквені ідентифікатори.

Модулі та пакети – `b`
Класи – `CapWords`
Винятки – `CapWords (+Error)`
Глобальні змінні , функції –
`lower_case_with_underscores`
Константи – `UPPERCASE`,
`UPPERCASE_WITH_UNDERSCORES`

PEP 8 – Стилі імен Python

Загальні рекомендації:

Аргументи функцій (методів) - завжди використовуйте `self` як перший аргумент методу екземпляру об'єкта, - завжди використовуйте `cls` як перший аргумент методу класу.

Завжди дотримуйтеся прийнятого Python однакового стилю (на базі PEP). Особливо при колективній роботі над проектом.

Посилання

- https://docs.python.org/3/library/exceptions.html
- https://pep8.ru/doc/pep8/
- https://github.com/python/peps/blob/master/pep-0008.txt

Контрольні питання

- Визначте поняття виняткової ситуації при виконанні операторів, надайте приклади виняткових ситуацій.
- Наведіть оператори обробки виняткових ситуацій, визначте порядок обробки винятку операторами try … except
- Надайте мету та визначте порядок використання оператора генерації виняткових ситуацій raise, надайте приклади.
- Надайте основні положення стилю коду Python (PEP-8)

The END
Лекція 11

ЛЕКЦІЯ 12. ПАКЕТИ #1

1. Вбудовані модулі (sys, math, random, copy)
2. Пакети NumPY : SciPY

Вбудовані модулі Python

Стандартний інтерпретатор Python має низку вбудованих модулів, що забезпечують взаємодію з оточенням.

Системні:

- **sys** – забезпечує доступ до системно-залежних параметрів та функцій.
- **string** – загальні операції з рядками.
- **datatime** – операції обробки часу і дати.
- **calendar** – функції роботи з календарем.
- **********

Математичні:

- **math (cmath)** – математичні функції (комплексні).
- **statistic** – функції математичної статистики.
- **random** – генерація випадкових чисел
- **********

Повний перелік дивись: https://docs.python.org/3/py-modindex.html

Вбудований модуль Sys

Модуль забезпечує доступ до змінних, що використовуються або підтримуються інтерпретатором, та до функцій, які сильно взаємодіють з інтерпретатором (> 100 функцій).

Типові функцій:

- `sys.prefix` - директорія встановлення інтерпретатору python.
- `sys.path` - список шляхів пошуку модулів.
- `sys.setrecursionlimit` – встановити максимальну глибину рекурсії.
- `sys.__stdin__, sys.__stdout__,` - значення потоків вводу, виводу.
- `sys.version` - версія python.
- `sys.version_info` - кортеж з п'яти компонентів номеру версії.
- `sys.platform` - інформація про операційну систему.

Посилання https://docs.python.org/3.8/library/sys.html

Приклади: EXAMPL_LEC_12_PYTHON_12_1_Sys_Math.ipynb

Вбудований модуль Math

Модуль забезпечує доступ до математичних функцій, визначених стандартом мови С.
Дійсні числа: `math`
Комплексні числа: `cmath`

Групи функцій:

- **Степенні і логарифмічні функції.**
- **Тригонометричні функції.**
- **Гіперболічні функції.**
- **Спеціальні функції.**
- **Константи.**
- **Чисельне - теоретичні функції.**

Посилання https://docs.python.org/3.8/library/math.html

Приклади: EXAMPL_LEC_13_PYTHON_12_1_Sys_Math.ipynb

Пакети: NumPy, SciPy

NumPy (нум пай) – n-вимірні масиви
https://numpy.org/
SciPy (сай пай) – наукові обчислення
https://scipy.org/
SymPy (сім пай) – символічні обчислення
https://www.sympy.org
Matplotlib - 2D графіка
https://matplotlib.org/

NumPy

Відкрита бібліотека (пакет) розширення Python для підтримки великих багатовимірних масивів та матриць та виконання операцій з ними.

Модулі:

- `np.emath` – математичні функції,
- `np.random` – випадкові функції,
- `np.fft` – дискретне перетворення Фур'є,
- `np.linalg` – лінійна алгебра,
- `np.matlib` – матричні операції.

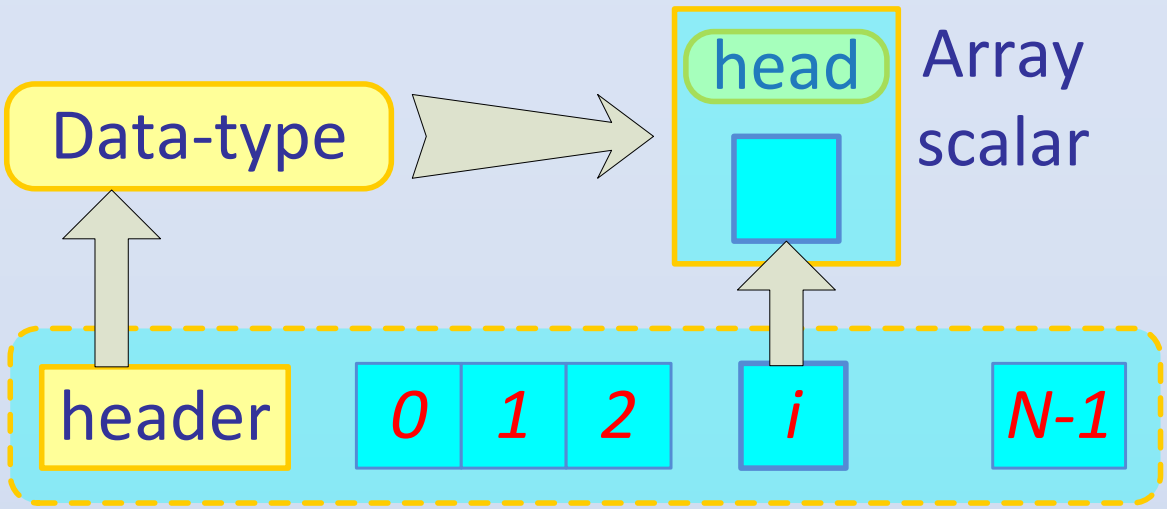
Функції:

- - сортування,
- - поліноми,
- - статистичні,
- - фінансові,
- - .....

Посилання https://uk.wikipedia.org/wiki/NumPy

Посилання https://docs.scipy.org/doc/numpy/reference/index.html

NumPy



ndarray – колекція елементів одного типу.

header – вбудований об'єкт опису масиву. Спосіб інтерпретації кожного елемента в масиві визначається окремим об'єктом – **data-type**. Крім основних типів (int, float, …), об'єкти типів даних також можуть представляти структури даних.
Вибраний елемент (за індексом) є об'єктом Python – array_scalar, вбудований у NumPy.

NumPy. Типи елементів

Базові array scalar типи:

- `int_` – цілочисловий тип,
- `float_` – тип з рухомою комою,
- `complex_` – комплексний тип,
- `bytes_` – байт тип,
- `unicode_` – символи юнікоду,
- `bool_` – логічний тип.

за замовчуванням `float_`

Варіації array scalar типів:

`int8`	`uint8`	`float8`	`complex64`	`bool8`
`int16`	`uint16`	`float16`	`complex128`	
`int32`	`uint32`	`float32`	`complex192`	
`int64`	`uint64`	`float64`	`complex256`	
		`float128`		

NumPy. Створення

5 базових механізмів створення масиву

- Внутрішній (arange, ones, …).
- Перетворення з інших структур Python (list, …).
- Зчитування з файлу .
- Формування з послідовності (strings, …).
- З використанням спеціальних функцій (random,…).

Внутрішній механізм:

- `array( object [,dtype, copy, order,…])`
- `empty(shape[,dtype,order])`- пустий масив
- `ones(shape[,dtype, order])`- масив 1-ць
- `zeros(shape[,dtype, order])`- масив 0-ів
- `full(shape,fill_value[,dtype, order])`- масив заповнений значеннями `fill_value`

`shape` – кортеж, що визначає розмірність,
`dtype` – тип елементу,
`order` – порядок збереження.

NumPy. Математичні операції

Функція	Повертає
negative(x1, / [, ...])	Поелементно -X1
positive (x1, / [, ...])	Поелементно +X1
add(x1,x2, / [, ...])	Поелементне X1 + X2
subtract(x1,x2, / [, ...])	Поелементне X1 - X2
multiply(x1,x2, / [, ...])	Поелементне X1 * X2
divide(x1,x2, / [, ...])	Поелементне X1 / X2
reminder(x1,x2, / [, ...])	Поелементне X1 % X2
mod(x1,x2, / [, ...])	Поелементне X1 % X2
**********	
sign(x, / [, ...])	Поелементне знак X
power(x1,x2, / [, ...])	Поелементне X1 ** X2

NumPy. Математичні функції

Функція	Повертає
exp(x, /[, ...])	Поелементна експонента
log(x, /[, ...])	Поелементний логарифм
sqrt(x, /[, ...])	Поелементний корінь
gcd(x1,x2, / [, ...])	Поелементний **НОД**
ТИНОНОМЕТРИЧНІ	
sin(x, /[, ...])	Поелементне ***sin()***
asin(x, /[, ...])	Поелементне ***asin()***
ГИПЕРБОЛІЧНІ	
sinh(x, /[, ...])	Поелементне ***sinh()***
asinh(x, /[, ...])	Поелементне ***asinh()***
ПЕРЕТВОРЕННЯ	
deg2rad(x, /[, ...])	Поелементне *град -> рад*
rad2deg(x, / [, ...])	Поелементне *рад -> град*

NumPy. Порівняння, логіка

Функція	Повертає
greater(x1,x2, / [, ...])	Поелементне >
greater_equal(x1,x2, / [, ...])	Поелементне >=
less_equal(x1,x2, / [, ...])	Поелементне <
not_equal(x1,x2, / [, ...])	Поелементне <=
equal(x1,x2, / [, ...])	Поелементне ==
greater(x1,x2, / [, ...])	Поелементне !=
logical_and(x1,x2, / [, ...])	Поелементне *and*
logical_or(x1,x2, / [, ...])	Поелементне *or*
logical_xor(x1,x2, / [, ...])	Поелементне *xor*
logical_not(x, / [, ...])	Поелементне *not*

NumPy. Модуль random

Функція	Повертає
Integers (low [, high, size, dtype, endpoint])	Випадкові цілі від *low* до *high* (виключно), або коли *endpoint*=**True**, до *high* (включно).
random ([size, dtype, out])	Випадкові з рухомою комою в інтервалі [0.0, 1.0).
bytes(length)	Випадкову послідовність байтів
Більш 30 функцій розподілення :	
normal([loc, scale, size])	Нормальне розподілення

https://docs.scipy.org/doc/numpy/reference/random/generator.html

NumPy. Статистика ….

Функція	Повертає
median(a[, axis, …])	Медіана масиву
average(a[, axis, …])	Середня арифметична
mean (a[, axis, …])	Мода масиву
std (a[, axis, …])	Стандартне відхилення
corcoef (x[, y, rowar, bias, ddof])	Коефіцієнти кореляції
correlate (x, v[, mode])	Взаємна кореляція двох масивів
histogram (a,[,bins,range, normed, …])	Гістограма розподілу значень в масиві

NumPy. Сортування ….

Функція	Повертає
sort(a[, axis, kind, order])	Отсортовану копію **a**
*********	
argmax(a[, axis, kind, order])	Індекс максимального елемента **A**
argmin(a[, axis, kind, order])	Індекс мінімального елемента **A**
where(condition, [x,y])	Елемент **x** або **y** , враховуючи умову **condition**
count_nonzero (a, [, axis])	Кількість не нульових елементів масиву **a**

NumPy. Модуль linalg

Функція	Повертає
Dot(a,b[, out])	Скалярне множення двох масивів **a b**
matmul([x1,x2, /[, out, casting, order, …])	Множення матриць **x1 x2**
linalg.matrix_power(a,n)	Степінь **n** матриці **a**
linalg.norm(x[,ord,axis,keep])	Норма масиву **x**
linalg.solve(a,b)	Рішення СЛАР **a*x = b**

https://docs.scipy.org/doc/numpy/reference/routines.linalg.html

Приклади: EXAMPL_LEC_13_PYTHON_12_2_NumPy.ipynb

SciPy

Відкрита бібліотека (пакет) високоякісних наукових та інженерних інструментів.

Модулі:

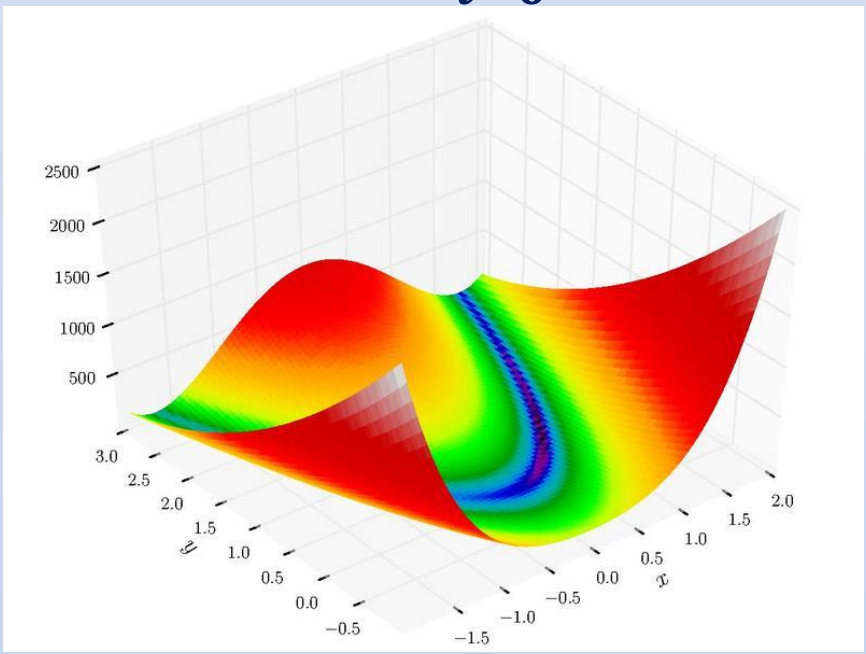
- `.constants` – фізичні константи,
- `.integrate` - інтегрування,
- `.optimize` - оптимізація,
- `.interpolate` - інтерполяція,
- `.fft` – перетворення Фурє,
- `.signal` - обробки сигналів,
- `.linalg` – лінійна алгебра,
- `.sprase` – розріджені матриці,
- `.spatial` – дерева, метрики,
- `.special` - спеціальні функції
- `.ndimage` - обробки зображень,
- `.stats` – статистичні функції.

Посилання https://docs.scipy.org/doc/scipy/reference/

SciPy (приклад)

Функція scipy.optimize.minimize() – безумовна мінімізація функцій багатьох змінних.
Тестова функція Розенброка:

$$f(x) = \sum_{i=0}^{n-2} [100 * \left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2]$$



Мінімум = 0, коли всі $x_i = 1$

https://ru.wikipedia.org/wiki/Тестовые_функции_для_оптимизации

Приклади: EXAMPL_LEC_13_PYTHON_12_3_SciPy.ipynb

SymPy

Відкрита бібліотека для символічної математики. Повнофункціональна комп'ютерною алгебра (CAS).

Основні модулі:

- `Polynomials` – поліноми,
- `Solving equations` – рішення рівнянь,
- `Combinatorics` - комбінаторика,
- `Discrete math` – дискретна математика,
- `Matrix` – матричні операції,
- `Geometry` – геометричні обчислення,
- `Physics` – фізичні обчислення,
- `Statistics` - статистика,
- `Cryptography` – криптографія.

... та багато іншого

Посилання https://www.sympy.org/en/index.html

Посилання

https://numpy.org/

Контрольні питання

- Визначте призначення і надайте призначення базових функцій модуля **sys**. Надайте приклади.
- Визначте призначення і надайте призначення базових функцій модуля **math**. Надайте приклади.
- Визначте призначення **numpy**. Поясніть принципи організації масивів в **numpy**.
- Пояснить основні механізми створення масивів в пакеті **numpy .** Надайте приклади.
- Надайте призначення модулів та функцій пакету **numpy**. Надайте приклади застосування.

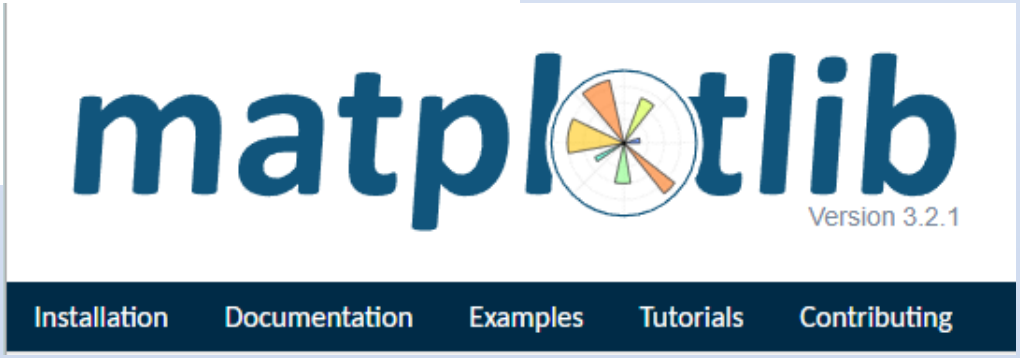
The END
Лекція 12

ЛЕКЦІЯ 13. ПАКЕТИ #2

Matpltlib

Пакет Matplotlib

https://scipy.org/



https://matplotlib.org/

Відкрита бібліотека для створення статичних, анімованих та інтерактивних візуалізацій на Python.

Графіка Matplotlib ≈ графіка MATLAB

Модулі Matplotlib

Основні модулі:

- `.pyplot` – функції побудови графіків в простих випадках,
- `.axes` – розвинуті функції побудови графіків.

та багато інших ……

- `.colors` – перетворення та управління кольорами,
- `.mathtext` – відображення виразів TeX та відображення на графіках,
- `.artist` – елементи відображення даних на графіках,
- `.image` – підтримка роботи з зображеннями,
- `.backend` – примітиви для UI (для використання результатів matplotlib,
- ……
- ~~`.pylab – (підтримка MATLAB)`~~ NO

Matplotlib: Загальна ідея

Ієрархія об'єктів:

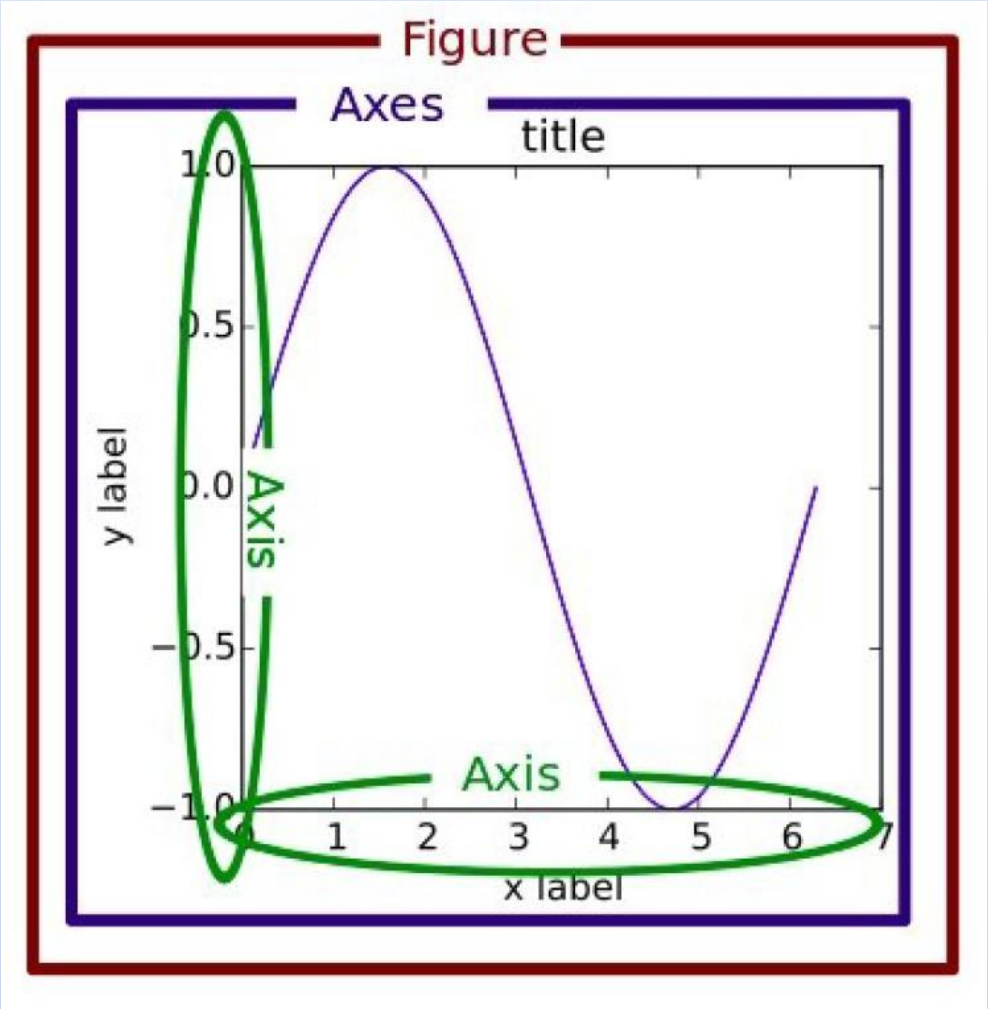


Figure – об'єкт, що містить усі елементи графіку.

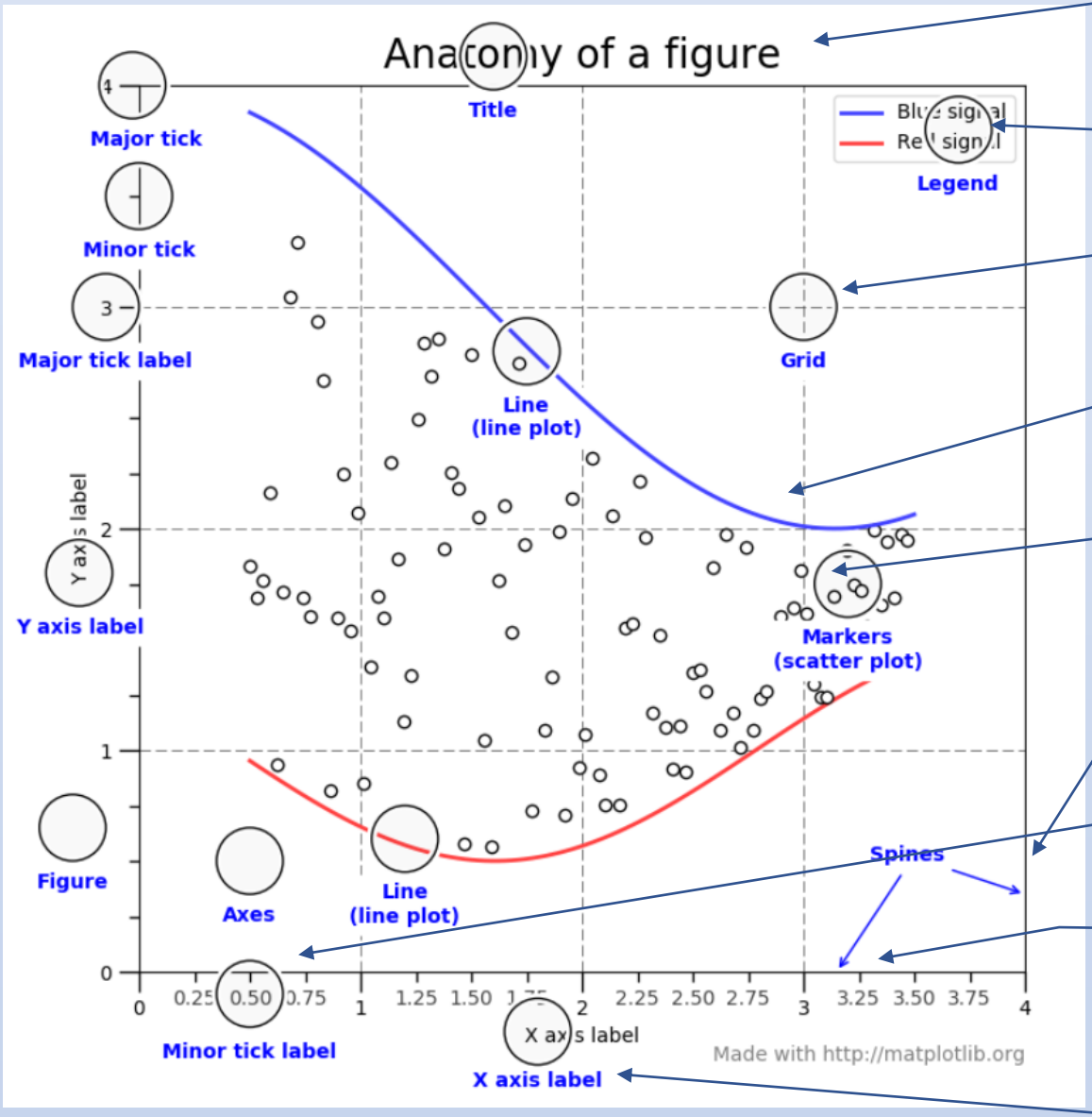
Axes – область малювання (елементарний графік).

Axis – координатні осі графіку.

Artist – клас усіх елементів графіку (всі елементи що відображуються – підклас Artist)

https://matplotlib.org/tutorials/introductory/usage.html

Matplotlib: Загальна ідея



Назва

Легенда

Сітка

Лінія

Маркери

Обрамлення

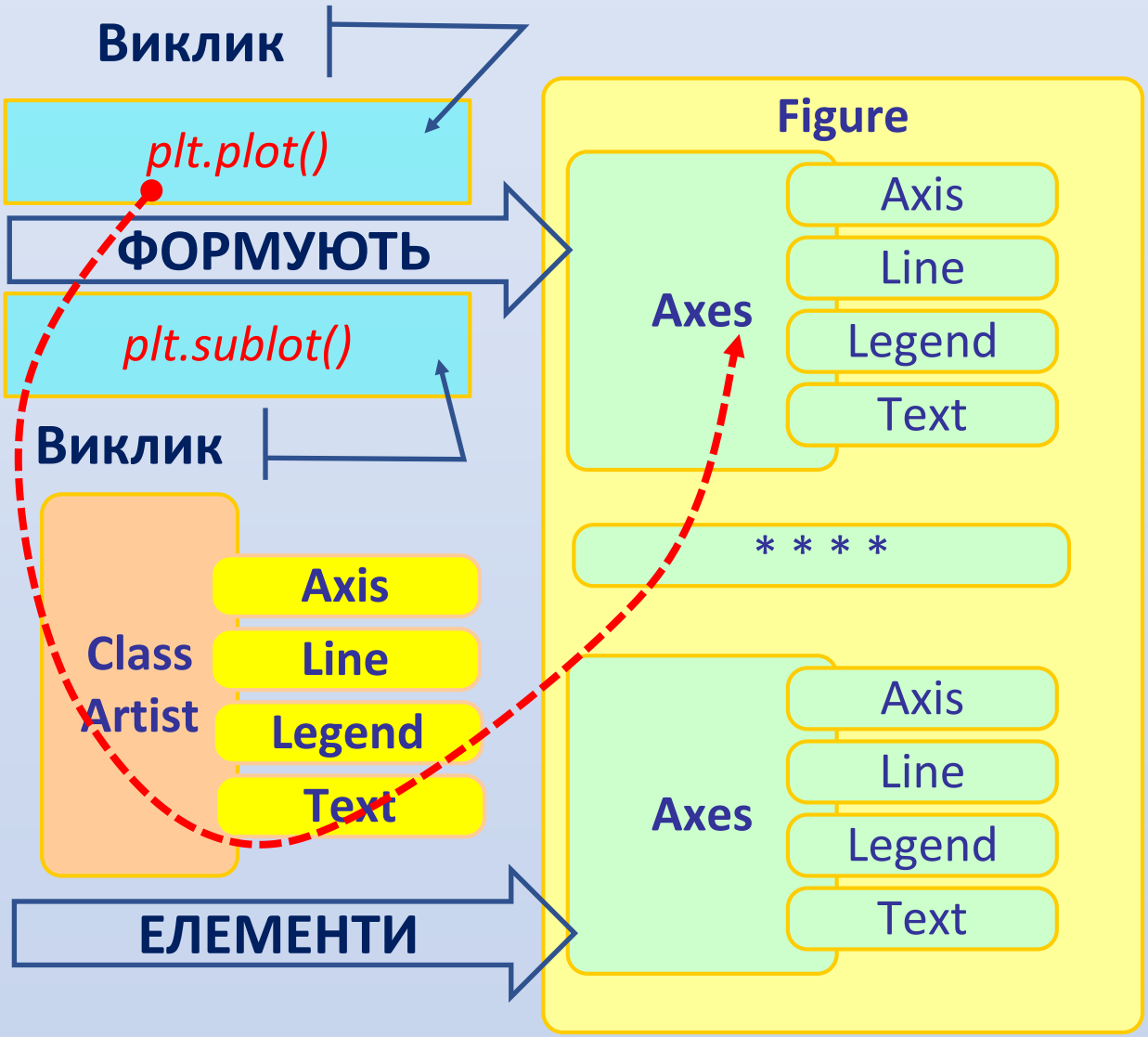
Позначки

Вісь X

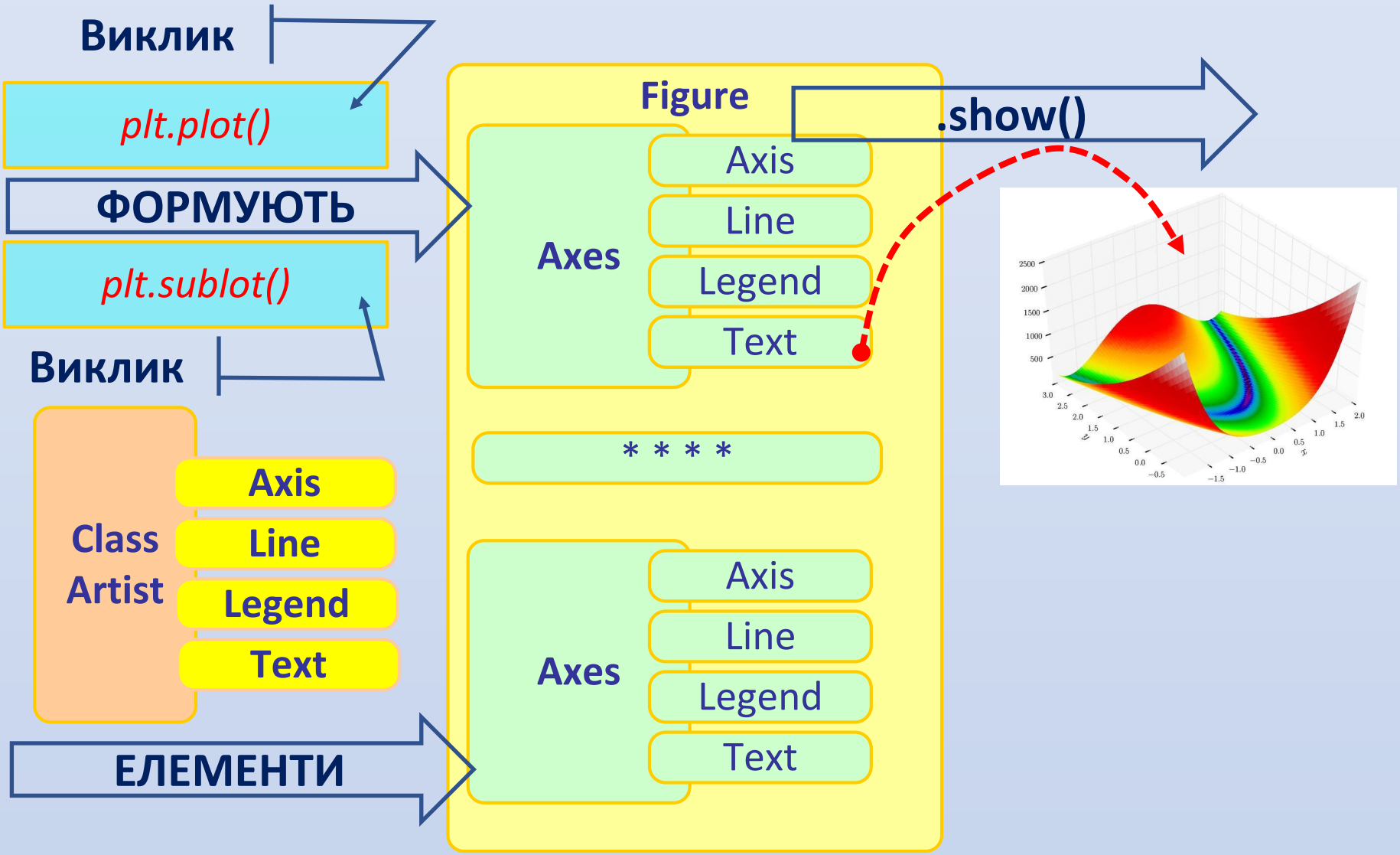
Назва осі X

https://matplotlib.org/tutorials/introductory/usage.html

Matplotlib: Загальна ідея



Matplotlib: Загальна ідея



Matplotlib: Загальна ідея

Базові типи графіків:

- `Line plot` – лінійні графіки.
- `Scatter plot` – діаграми розсіювання.
- `Bar chart` – стовпчасті діаграми.
- `Histogram` – гістограми.
- `Steam plot` – діаграми виду «стовбур-листя».
- `Contour plot` – контурні графіки.
- `quiver` – поля градієнтів.
- `Spectrogram` – спектральні діаграми.

- **********

Pyplot методи

Функція	Повертає
.scatter	Маркер
.plot	Лінія
.bar	Стовпчаста діаграма
.hist	Гістограма
.pie	Кругова діаграма
.contour	Ізолінії
**********	
.quiver()	Векторне поле
**********	
.show()	Вивід сформованого графіка
.imshow()	Вивід цифрового зображення

Pyplot.bar()

```
bar(x, height, width=0.8,bottom=None, *,
    align='center',data=None, **kwargs)
```

x – X координати стовпчиків (scalar sequence),
height – висота стовпчиків (scalar sequence),
……

Pyplot.plot()

```
plot(*args,scalex=True, scaley=True,
    data=None, **kwargs)
```

Типовий виклик:

```
Plot([x], y,[fmt], y2,[fmt2], … ,**kwargs)
```

X - X координати
Y - Y координати
[fmt] – рядок, що визначає форматування
`'[marker][line][colors]'`

Pyplot.scatter()

```
bar(x,y,s=None, c=None, marker=None,…,
    data=None, **kwargs)
```

x,y – координати точок ()
S – розмір точок,
C – колір точок.

……

Pyplot.pie()

```
pie(x, explode=None, labels=None,
colors=None,   data=None)
```

X - «частка пірогу». Частка = x/sum(x)

Приклад: EXAMPL_LEC_14_PYTHON_13_1_MatPlotLib.ipynb
Приклад: EXAMPL_LEC_14_PYTHON_13_2_MatPlotLib.ipynb

Властивості графічних елементів

Властивість	Пояснення
Color/colors/c	Колір
Linewidth	Товщина лінії
Linestile	Тип лінії
Alpha	Ступень прозорості (0 -> 1)
Fontsize	Розмір шрифту
Marker	Тип маркеру
S	Розмір маркеру
Rotation	Поворот рядку
**********	

Matplotlib: frontend : backend

“Frontend" - це код, створений програмістом, результат виконання якого є деякий графік, що який повинен бути візуалізовано користувачеві.

Matplotlib орієнтовано на різні засоби візуалізації результатів, і кожна з цих можливостей називається **“backend”:**

1. «Inline» - безпосередньо, як результат виконання коду в оболонці (consol, spyder, …)

2. «Interactive» - інтерфейс користувача (для використання в pygtk, tkinter, qt, ….

3. «Hardkopy» - для створення файлів зображень PNG, SVG, PDF, … ("неінтерактивні засоби").

Matplotlib: backend Qt

Початковий вигляд

Зміщення

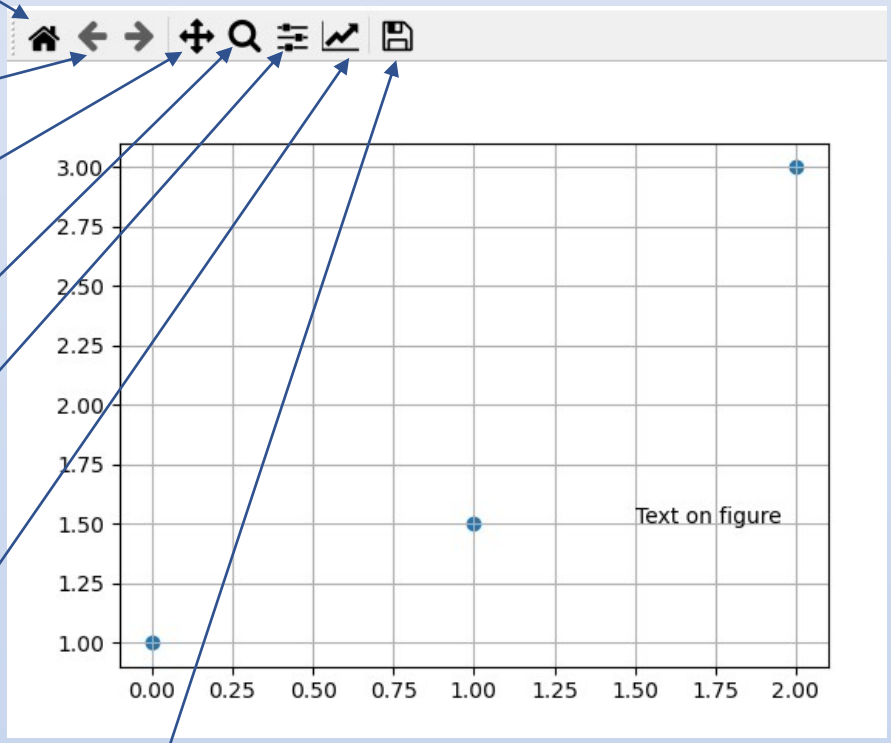
Панорамування

Масштаб

Форматування

Налаштування

Збереження



Toolkit: Matplot3D

Matplot3D - інструмент побудови тривимірних графіків (точкових, лінії, сітки). Будує 2D проекцію тривимірної сцени.

1. **Line plots**
2. **Scatter plots**
3. **Wireframe plots**
4. **Polygons plots**

*****

```
axes3D.plot(self, xs, ys, *args, zdir='z',
    **kwargs)
```

xs - X координати точок (1D масив)
ys - Y координати точок (1D масив)
zs - Z координати точок (1D масив)
Zdir - напрям відрисовки

https://matplotlib.org/mpl_toolkits/mplot3d/tutorial.html

Приклад: EXAMPL_LEC_14_PYTHON_13_3_MatPlot3D.ipynb

Посилання

- https://matplotlib.org/
- https://github.com/rougier/matplotlib-tutorial
- https://github.com/whitehorn/Scientific_graphics_in_python

Контрольні питання

- Визначте призначення бібліотеки **matplotlib**, надайте опис базових модулів пакету, поясніть механізм процесу створення графіка.
- Визначте призначення функції **pyplot.plot()**. Надайте приклади використання.
- Визначте призначення функції **pyplot.bar()**. Надайте приклади використання.
- Визначте призначення функції **pyplot.pie()**. Надайте приклади використання.
- Визначте призначення функції **pyplot.scatter()**. Надайте приклади використання.

The END
Лекція 13

ЛЕКЦІЯ 14. ПАКЕТИ #3

1. Пакет Qt5
2. Пакет PyQt5

Qt5

Qt (к'ют) — інструментарій розробки ПЗ мовою C++.

- Крос – платформеність: Linux, Windows, MacOS, вбудовані пристрої.
- Строга ієрархія класів.
- Багатий вибір елементів керування.
- Класи підтримки ! Всього ….

Qt підтримує зв’язок з Ada, C#, Java, PHP, Ruby та **Python (PyQt, PySide).**

https://www.qt.io/

Qt5

Qt — має модулі класів, які можуть бути потрібні для розробки практично любого прикладного ПЗ:

- **QtCore – базові класи.**
- **QtGui – класи графічного інтерфейсу.**
- **QtWidgets – класи графічного інтерфейсу у вигляді віджетів.**
- **QtNetwork – класи мережевого програмування .**
- **QtMultimedia – класи роботи зі звуком, відео … .**
- **QtSql – класи роботи з базами даних .**
- **QtSvg – класи програмування векторної графіки.**
- **QtWebView – класи підтримки відображення веб контенту.**
- **Qt3D – класи підтримки програмування 3D графіки.**
- **. . . .**

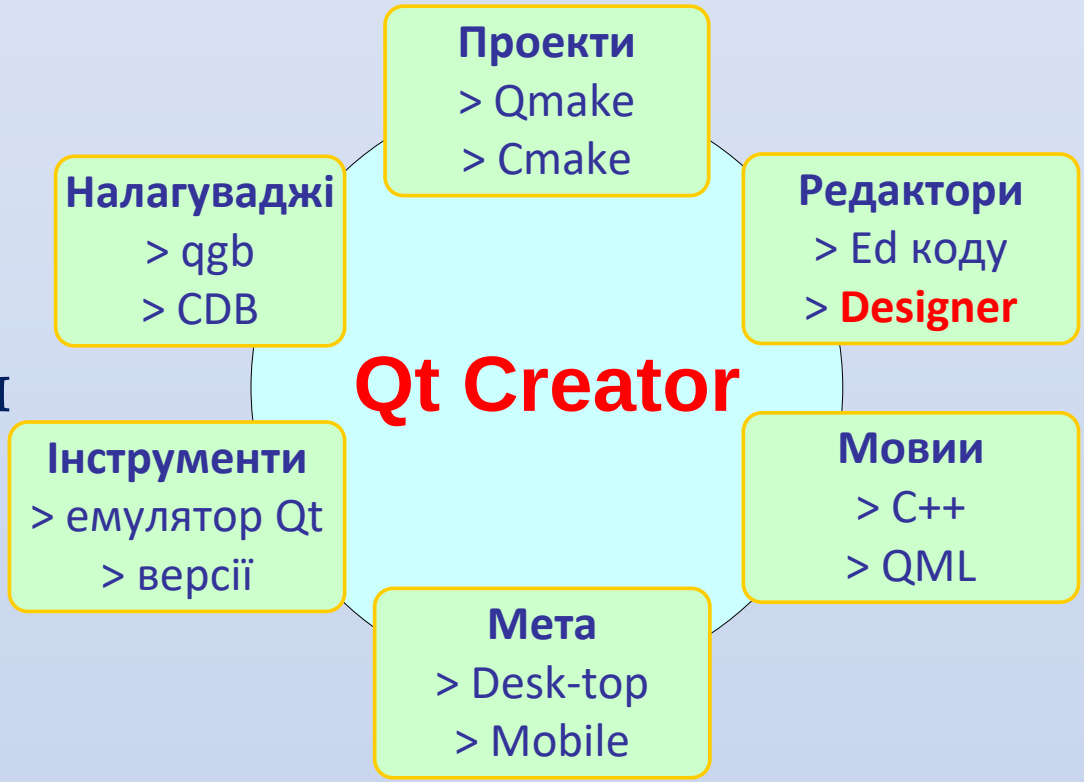
https://doc.qt.io/

Qt5 інструментарій

QtCreator — кросс-платформена відкрите IDE для розробки ПЗ на C, C ++ та QML.

Включає:

- **Графічний інтерфейси.**
- **Налагуваджі.**
- **Візуальні засоби розробки інтерфейсу.**
- **Емулятори.**



https://www.qt.io/

Qt5 інструментарій

QtDesigner — кросс-платформенне відкрите середовище для розробки графічних інтерфейсів (GUI) програм що використовує бібліотеку Qt. Входить до складу QtCreator.

Розроблений GUI зберігається в файл з розширенням ui, який підключається до створюваної програмі за допомогою спеціальних методів бібліотеки Qt.

Файл має xml-формат, і може, в разі необхідності, редагуватися в будь-якому текстовому редакторі.

https://doc.qt.io/archives/qt-4.8/designer-manual.html

Qt5: сигнали & слоти

Signals & Slot — спеціальний механізм генерації та обробки повідомлень.

Signal - спеціальний метод об'єкту, який генерує повідомлення. Причина генерації – внутрішня справа об'єкту.

Slot - спеціальний метод об'єкту, який приймає повідомлення.

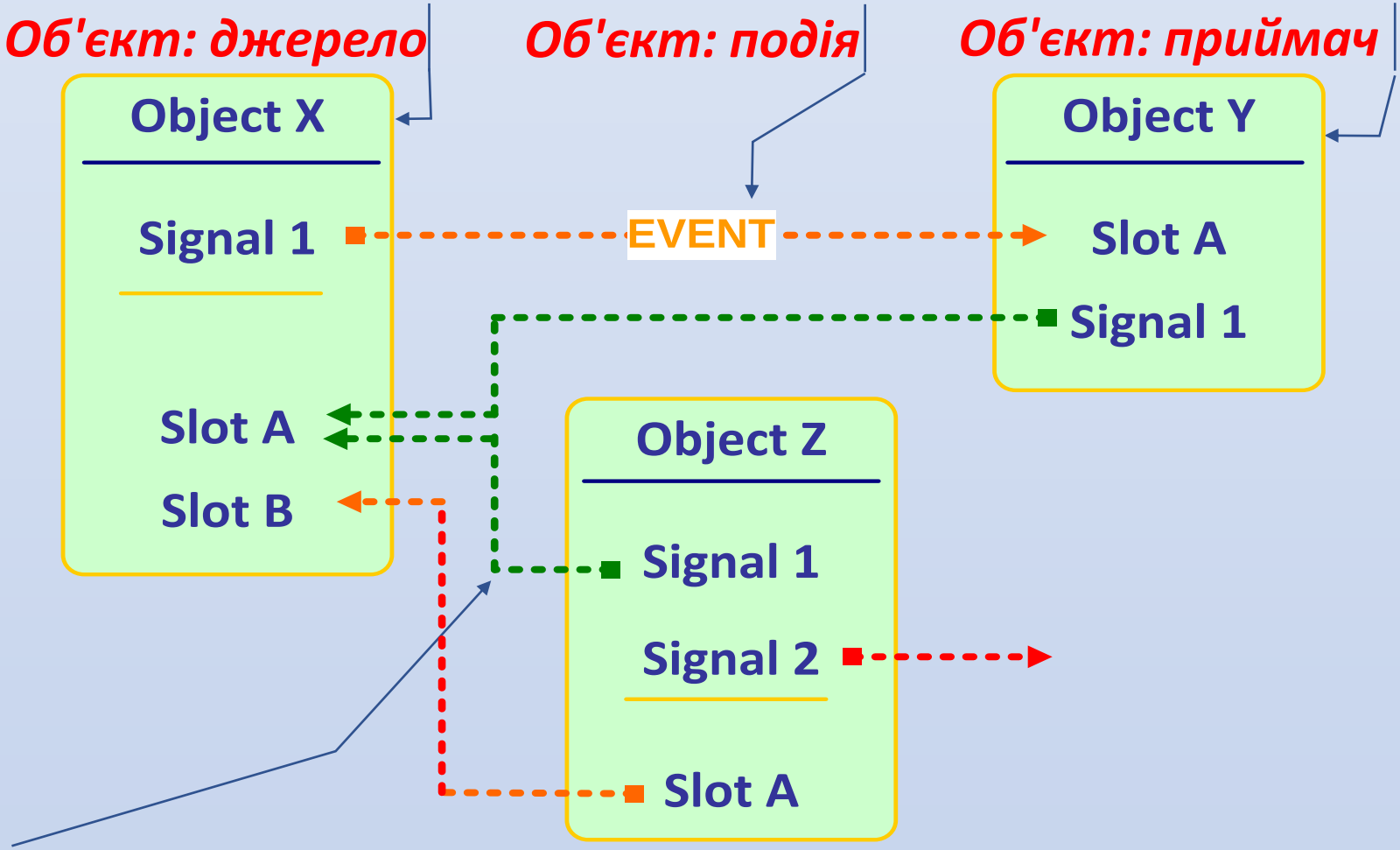
З'єднання сигналу та слоту виконується за допомогою методу

connect(*sender, signal, receiver, slot, type*)

Перед стандартній компіляцією проекту, його обробляє **спеціальний MOC (meta-object compiler)** компілятор, додаючи і підміняючи код для відповідності ISO C ++.

Qt5: сигнали & слоти

QT використовує ПОДІЙНО-ОРІЄНТОВАНУ Модель. Виклик exec_( ) – безкінечний цикл, який получає події та обробляє їх.



Connect(ObjectZ,signal1, ObjectX, slotA)

Qt5: сигнали & слоти

- Будь-який клас може мати необхідну кількість слотів і сигналів (практично необмiжну).
- Будь-який сигнал може підключатися до будь-якої кількості слотів і навпаки.
- Якщо сигнали та слоти є загальнодоступними властивостями, будь-який клас може з'єднуватися з ними коли завгодно.
- Кожен клас може відключити свій слот або сигнал у будь-який час, коли його більше не цікавлять події.
- Якщо клас знищено, він автоматично відключає всі його сигнали та слоти.

Qt5→PyQt5

PyQt5 — бібліотека, яка дозволяє використовувати Qt GUI з Python скриптів. Використовуючи PyQt5 з Python, можна створювати програми набагато швидше, не жертвуючи швидкість C ++.
PyQt5 поєднує в собі всі переваги Qt та Python.
PyQt5 посилається на останню версію 5 Qt.
PyQt5 розповсюджується за ліцензією GPL v3.
PyQt не включає копію Qt. Необхідно отримувати правильно ліцензовану копію Qt.
! Qt4 не підтримується.

https://pypi.org/project/PyQt6/
https://pypi.org/project/PyQt5/
https://www.riverbankcomputing.com/static/Docs/PyQt5/index.html

Модулі PyQt5

PyQt5 – включає велику кількість модулів (52):

	Класи …
QtCore	Ядро класів Qt (неграфічна функціональність)
QtGui	… спільні для графічних інтерфейсів віджетів та OpenGL
QtDesigner	… розширення Qt Designer за допомогою Python
QtWidgets	… створення класичних інтерфейсів Desk-top стилю
QtMultymedia	… крування мультимедійним контентом (камери, радіо, …)
QtDataVisualisation	… підтримки візуалізації даних у 3D
QtOpenGL	… візуалізації OpenGL у традиційних віджетах
Qt3DRander	… 2D та 3D-рендерінгу
Qt3DAnimation	…. Підтримки анімації

https://www.riverbankcomputing.com/static/Docs/PyQt5/module_index.html#ref-module-index

Модулі PyQt5

PyQt5 – включає велику кількість модулів (52):

	Класи …
QtCore	Ядро класів Qt (неграфічна функціональність)
QtGui	… спільні для графічних інтерфейсів віджетів та OpenGL
QtDesigner	… розширення Qt Designer за допомогою Python
QtWidgets	… створення класичних інтерфейсів Desk-top стилю
QtMultymedia	… крування мультимедійним контентом (камери, радіо, …)
QtDataVisualisation	… підтримки візуалізації даних у 3D
QtOpenGL	… візуалізації OpenGL у традиційних віджетах
Qt3DRander	… 2D та 3D-рендерінгу
Qt3DAnimation	…. Підтримки анімації

https://www.riverbankcomputing.com/static/Docs/PyQt5/module_index.html#ref-module-index

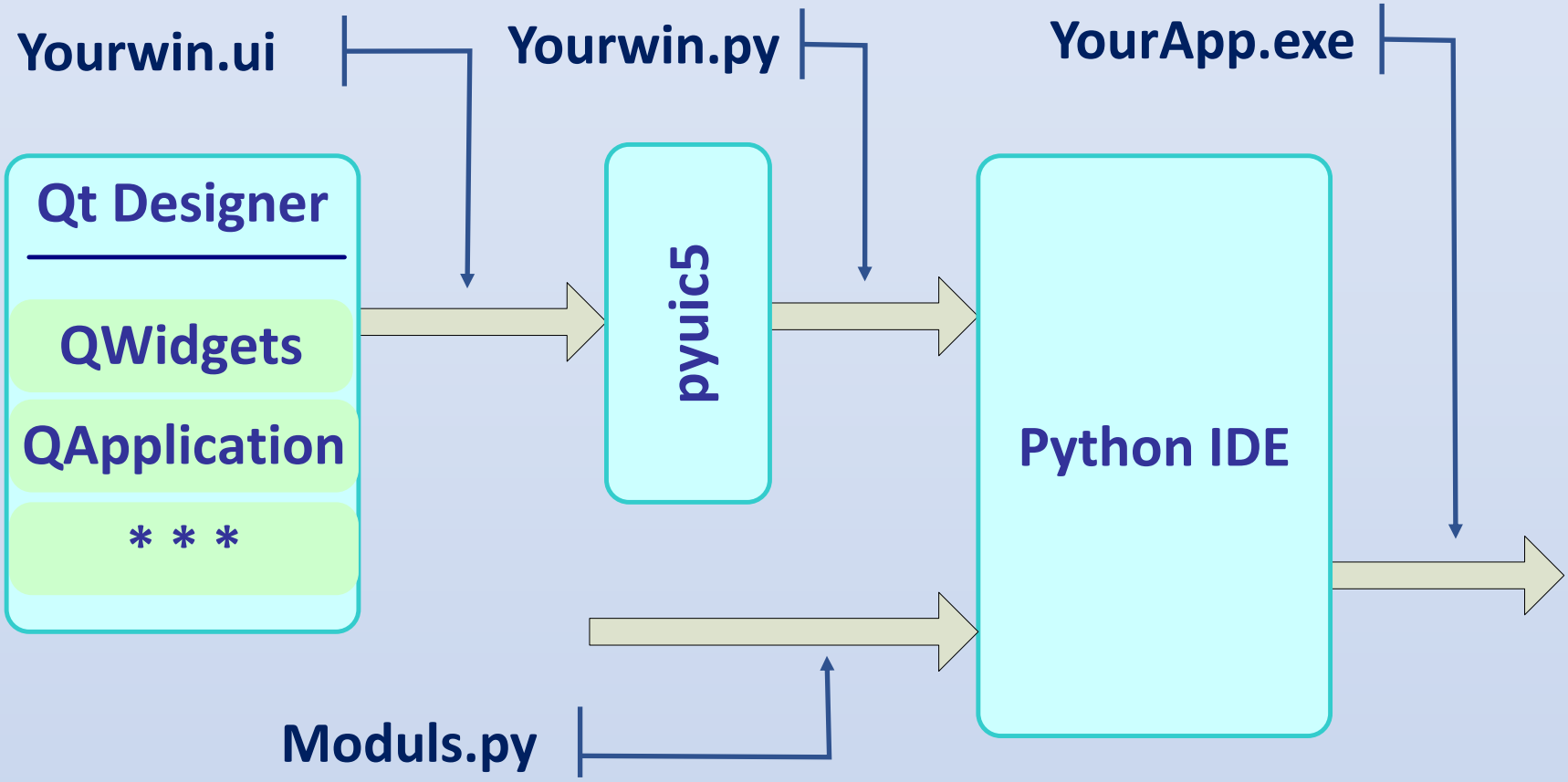
PyQt5: Qt Designer → модуль *.py

PyQt5 – містить ряд корисних утиліт:

- **pyuic5 відповідає утиліті Qt uic - перетворює інтерфейси на основі QtWidgets, створені за допомогою Qt Designer, в код Python.**

- **pyrcc5 відповідає утиліті Qt rcc - вбудовує довільні ресурси (наприклад, піктограми, зображення, файли перекладу), описані файлом збору ресурсів у модулі Python.**

Схема створення



TkInter

Tkinter (від *Tk interface*) - крос-платформна подієво-орієнтована графічна бібліотека на основі засобів Tk (широко поширена у світі GNU/Linux та інших UNIX-подібних систем, портована також і на Microsoft Windows).

Tkinter — вільне програмне забезпечення, яке розповсюджується під Python-ліцензією

Входить до стандартної бібліотеки Python.

Python interface to Tcl/Tk

https://docs.python.org/3/library/tkinter.html

Посилання

- https://doc.qt.io/qt-5/
- https://www.riverbankcomputing.com/static/Docs/PyQt5/introduction.html#pyqt5-components

Контрольні питання

- Визначте призначення пакету **Qt**, надайте опис базових модулів пакету, поясніть можливі механізми використання.
- Поясніть призначення механізму сигнал / слот , що використовується пакетом **Qt.**
- Визначте призначення пакету **PyQt**, надайте опис базових модулів пакету, поясніть можливі механізми використання.

The END
Лекція 14

ЛЕКЦІЯ 15
ТЕХНОЛОГІЇ

ЛЕКЦІЯ 15. ТЕХНОЛОГІЇ #1

1. Системи керування версіями.
2. Git. Стан та контроль файлів.
3. Git. Знімки.
4. Git. Робота з гілками.

VCS. Системи керування версіями

VCS (*Version Control System*) - програмний інструмент для керування версіями програмної інформації: початкового коду програми, скрипту, веб-сторінки, веб-сайту, тексту ...

Два класи:

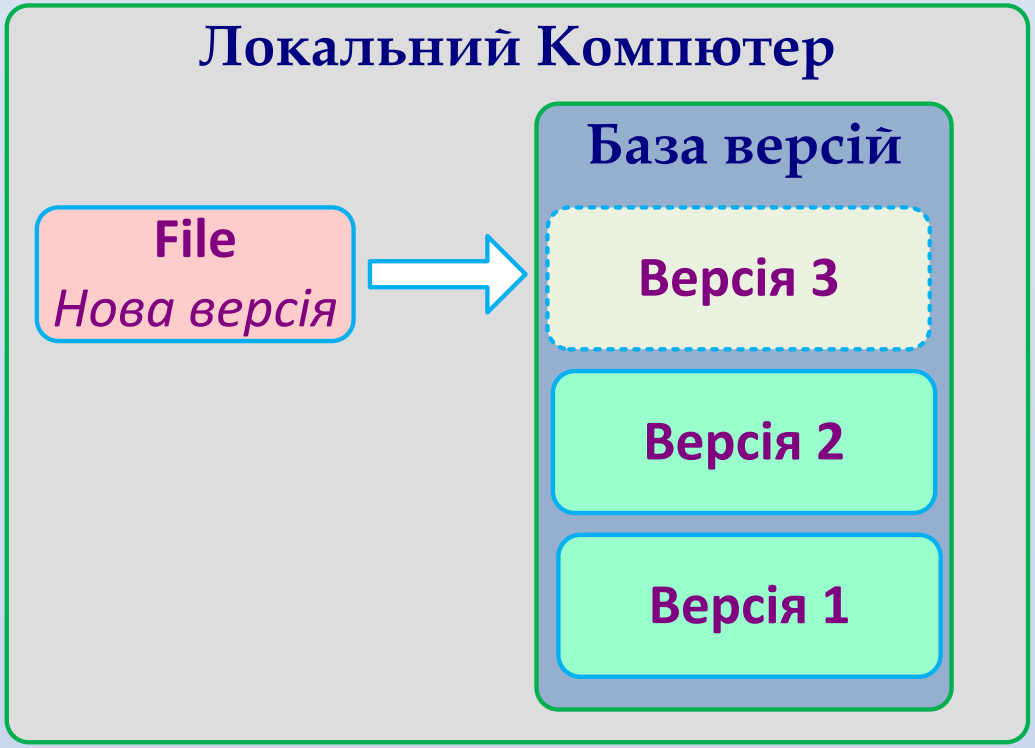
Централізовані – вся інформація зберігається в єдиному «серверному» сховищі. Обмін файлами відбувається через центральний сервер. Підтримується можливість створення та роботи з локальними робочими копіями.

Розподілені – використовує розподілену модель зберігання файлів, не потребує сервера: всі файли знаходяться на кожному з робочих комп'ютерів розробників.

VCS. Локальна система

«Наївний» підхід - копіювання файлів в окрему директорію.

Найпростіша VCS - зберігає відмінності між файлами в спеціальному форматі на диску. Може відтворити будь-який файл в будь-який момент часу.

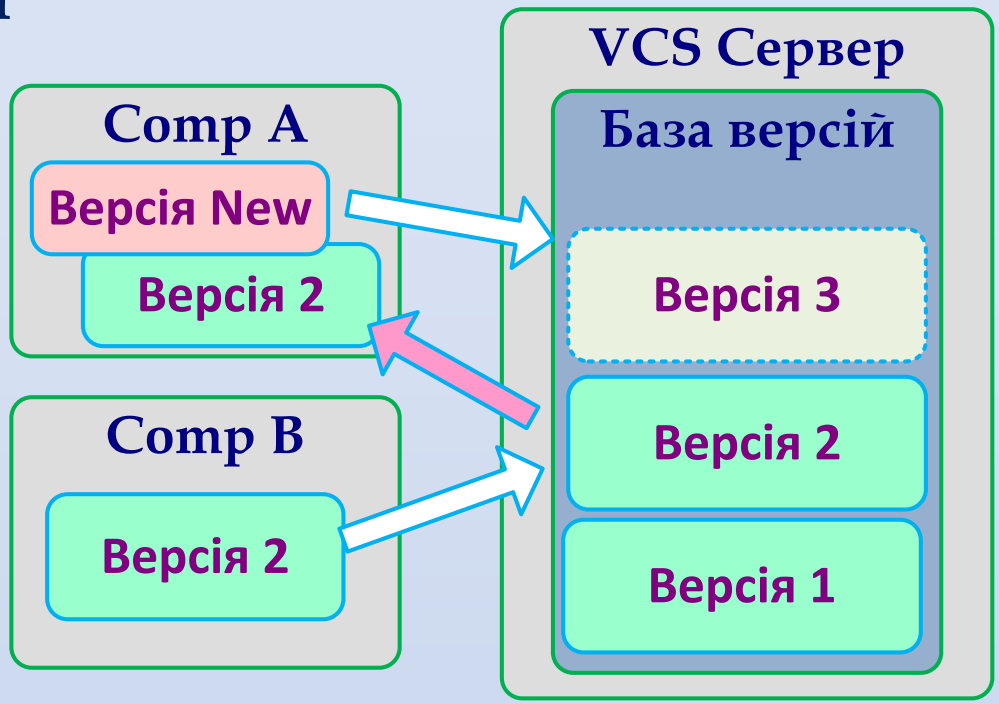


Revision Control System (RCS, 1985) - при зберіганні текстових файлів використовується алгоритм дельта-компресії, коли зберігається тільки перша версія і всі наступні зміни.

VCS. Централізована система

! Колективна розробка

Система має єдиний сервер, який містить всі версії файлів. Клієнти (розробники) отримують файли з центральної бази (репозиторію).



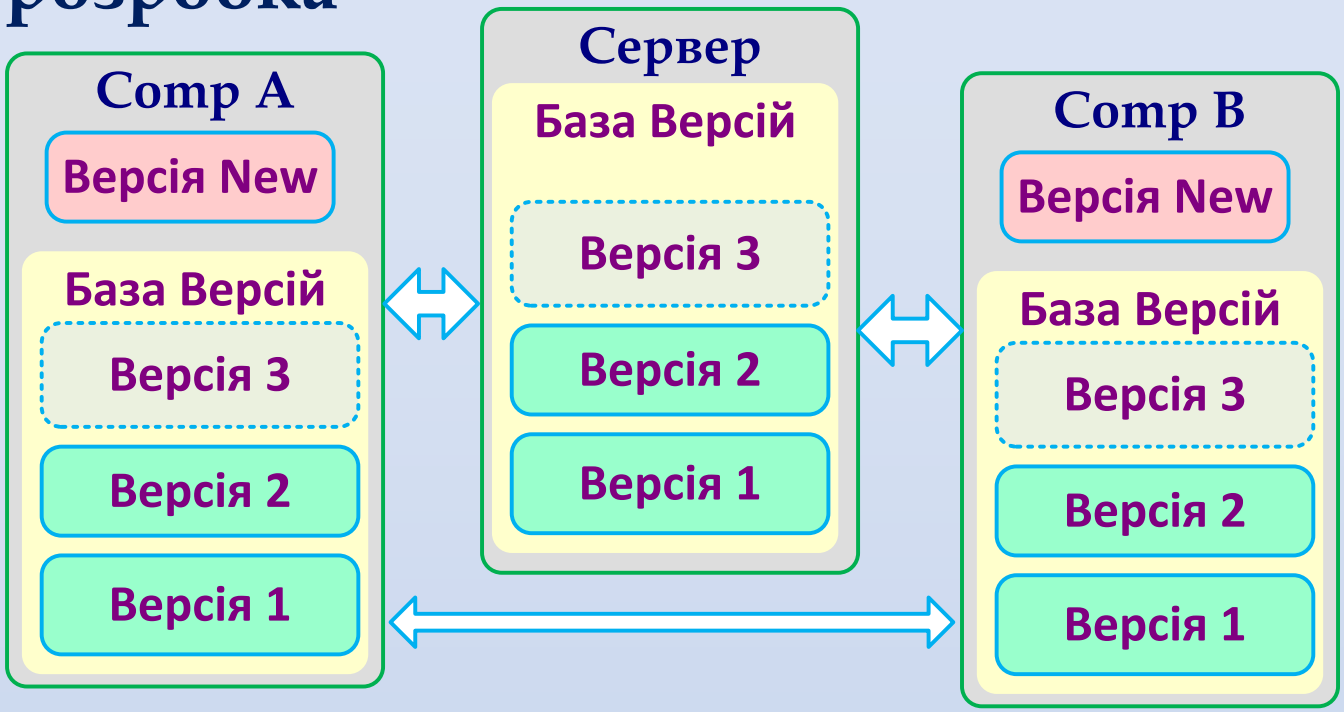
Декілька клієнтів працюють одночасно. Коли зміни приймаються номера версій змінених файлів автоматично збільшуються, і сервер записує коментар, дату і ім'я користувача в свій журнал.

CVS (Concurrent Versions System), 1990 – 2008
Perforce (P4), 1995 - ?
Subversion (SVN), 2004 →

VCS. Децентралізована система

! Колективна розробка

Локальні комп'ютери мають повну копію репозиторію



Системи підтримують взаємодію з декількома віддаленими репозиторіями, що дозволяє співпрацювати з різними групами розробників і мати одночасно декілька типів робочих процесів.

Git, 2005 --
Bazaar, 2005 – 2016
……..

Git

Розподілена система керування версіями файлів та спільної роботи. Розробник L.Torvalds (Linux).

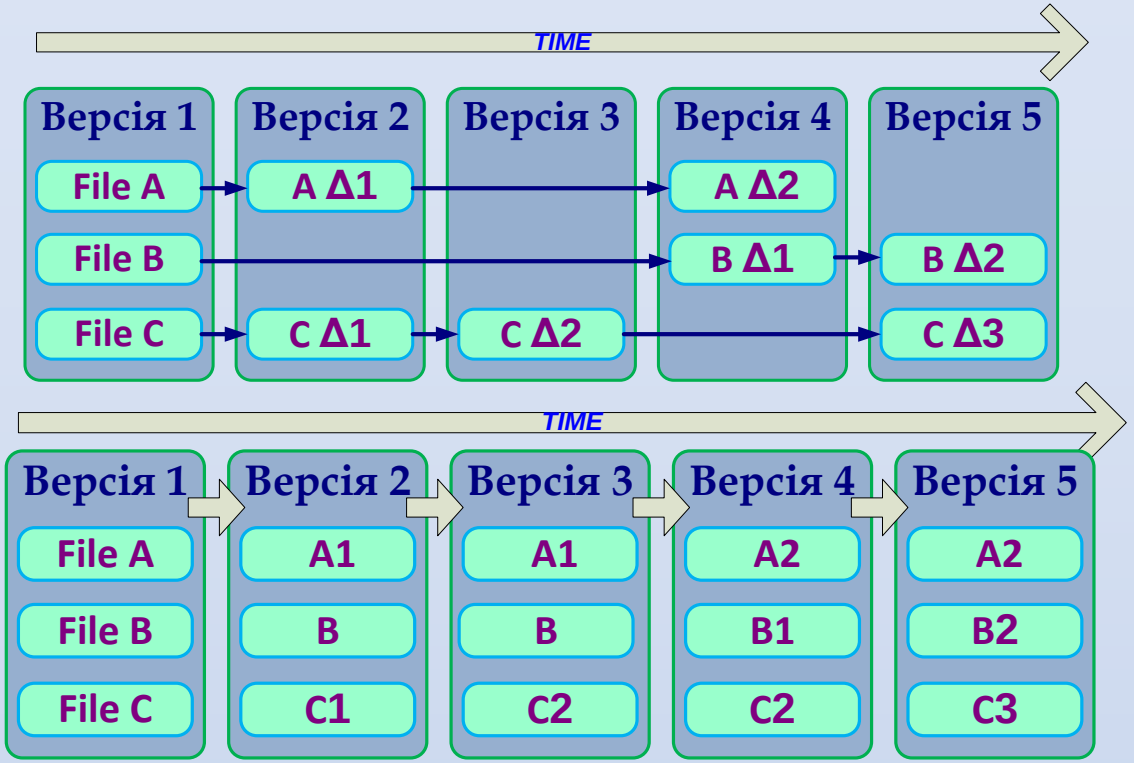
Git – *надійна, високопродуктивна децентралізована VCS*, надає гнучкі засоби нелінійної розробки, що базуються на відгалуженні і злитті гілок (тисячі паралельних гілок).
Цілісність історії та стійкість до змін забезпечується криптографічними методами, можлива прив'язка цифрових підписів.

https://en.wikipedia.org/wiki/Git

Git

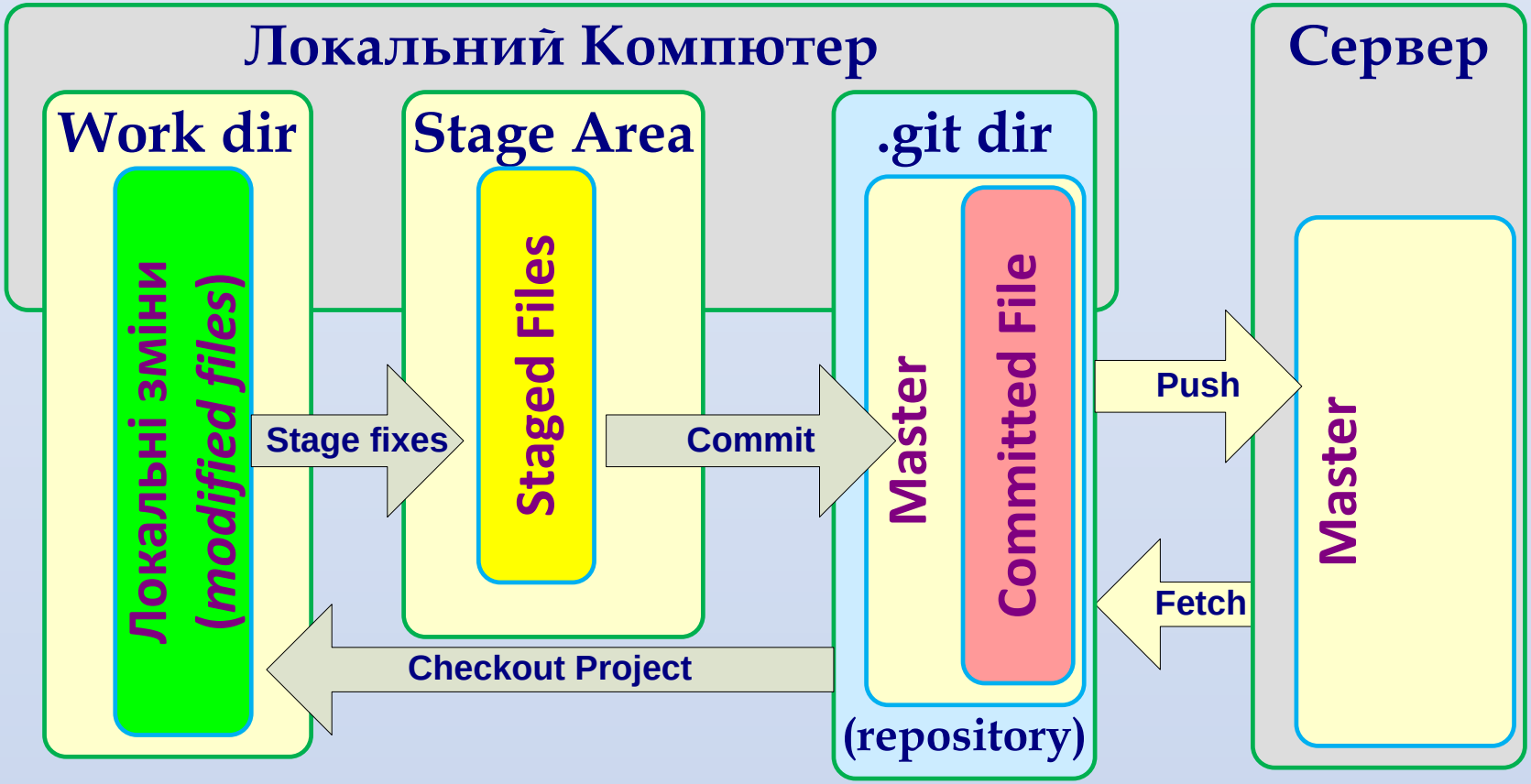
VCS –
дельта –контроль версій файлів.

Git –
Зберігання «знімків» (зліпків, *snapshots*) версій файлів.



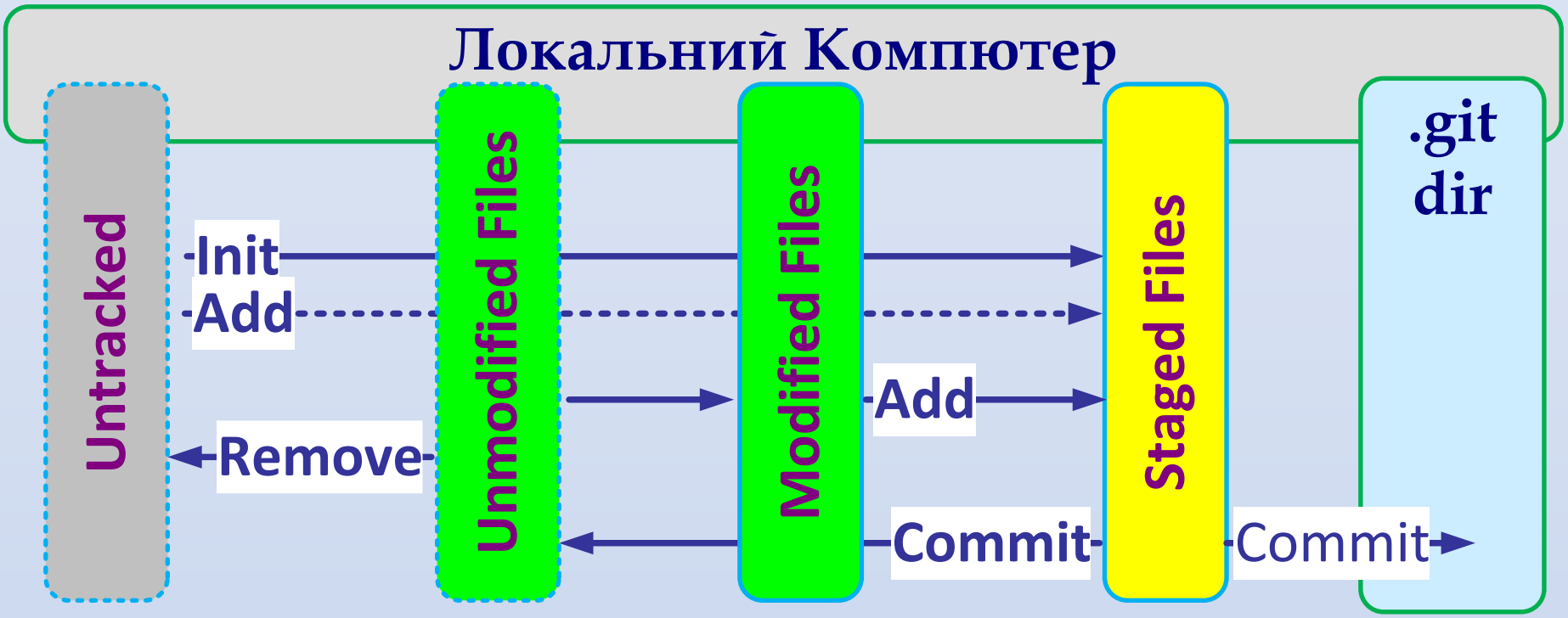
- **Нема необхідності з зв'язку з іншими комп'ютерами, повна історія на кожному локальному комп'ютері. Локальна робота.**
- **Перед збереженням, формується контрольна сума (хеш, SHA-1, 40 символів), за якою можна посилатися на «знімок».**
- **Неможливо змінити файл чи директорію так, щоб це було невідомо системі.**

Git. Стани файлів



- Змінений (modified) – до файлу внесені зміни.
- Індексований (staged) – змінений файл, що буде внесений до наступного знімку.
- Збережений (commited) – внесений до знімку.

Git. Контроль файлів



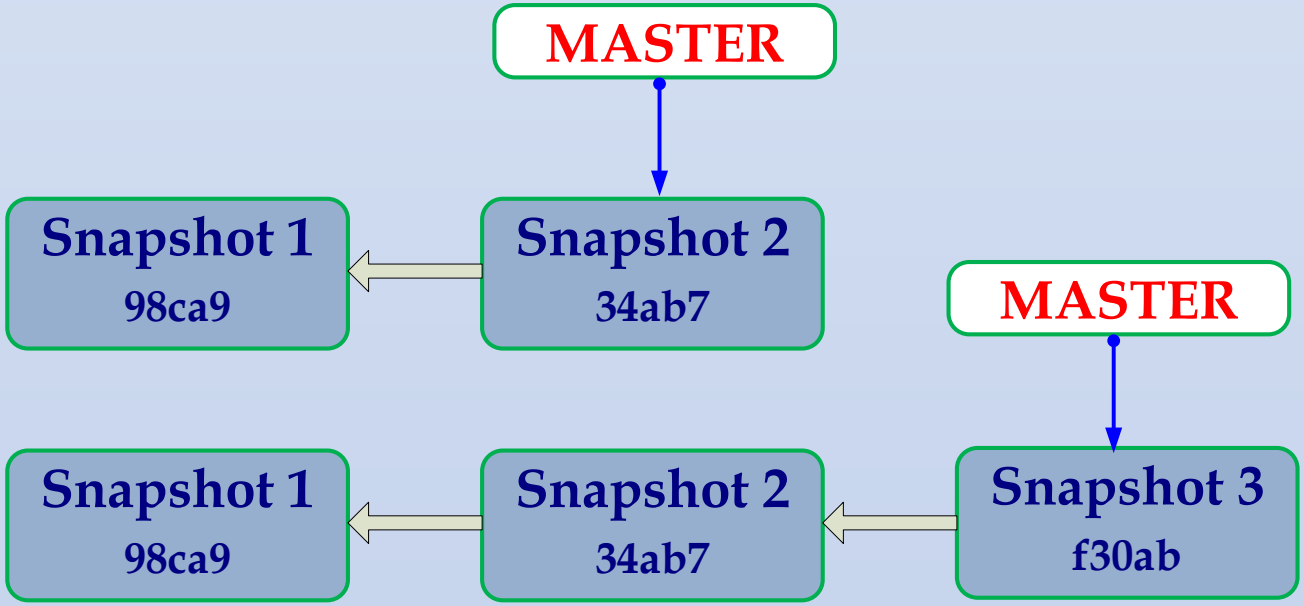
- *init* – створення локального (пустого) git репозиторію.
- *add* – додавання файлу до stage (індексація).
- *commit* – фіксує «знімок» в репозиторії.
- *log* – історія коммітів.

Git. Зберігання знімків

COMMIT → Git обчислює хеш кожної робочої директорії (файлу) та зберігає ці об'єкти дерева в сховищі Git. Потім Git створює об'єкт snapshot, що зберігає метадані та вказівник на корінь дерева проекту. Це дозволяє відтворити цей знімок, коли потрібно.

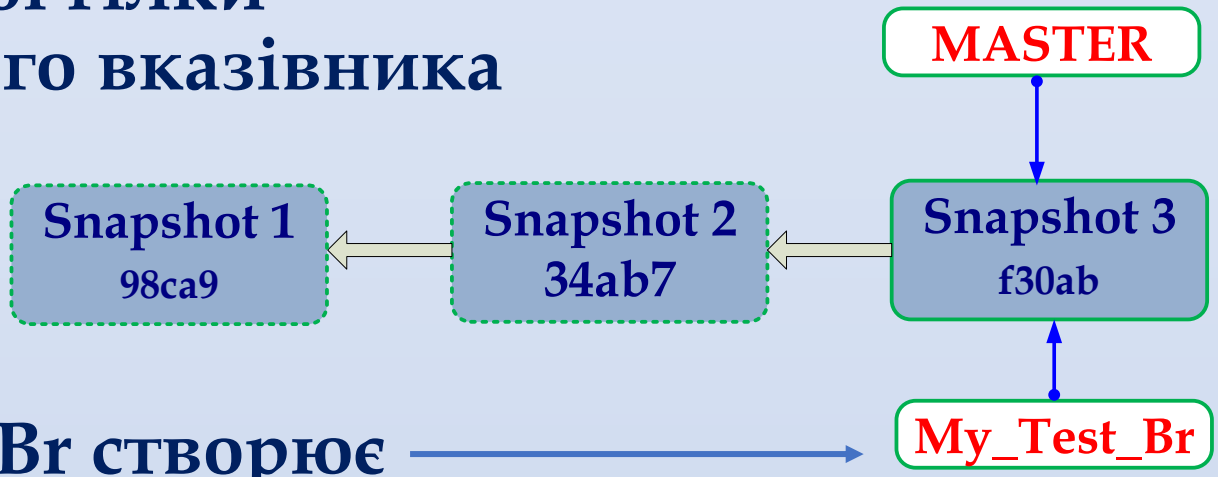
Git. Гілки (branches)

Гілка - вказівник, що може пересуватись та посилатись на одну з фіксацій (snapshots). За замовчуванням ім'ям першої гілки в Git є master. Коли виконується commit, вона переміщується вперед автоматично.



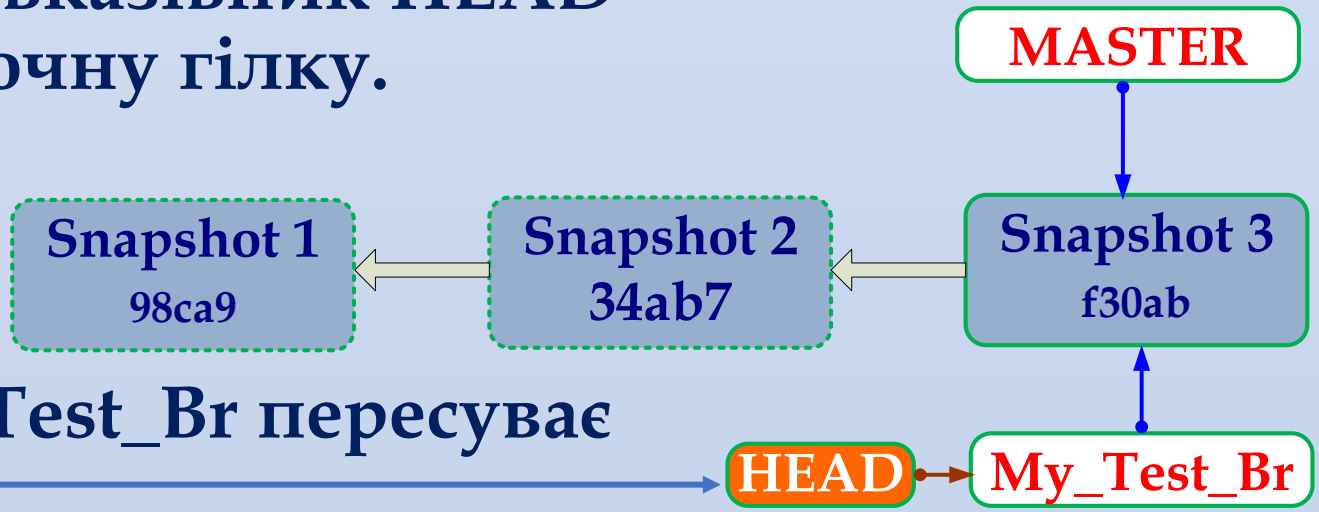
Git. Гілки (branches)

Створення нової гілки – створення нового вказівника



branch **My_Test_Br створює**

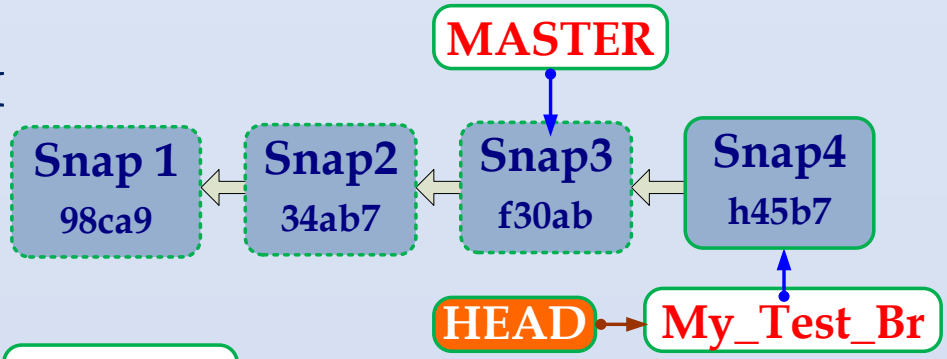
Спеціальний вказівник HEAD вказує на поточну гілку.



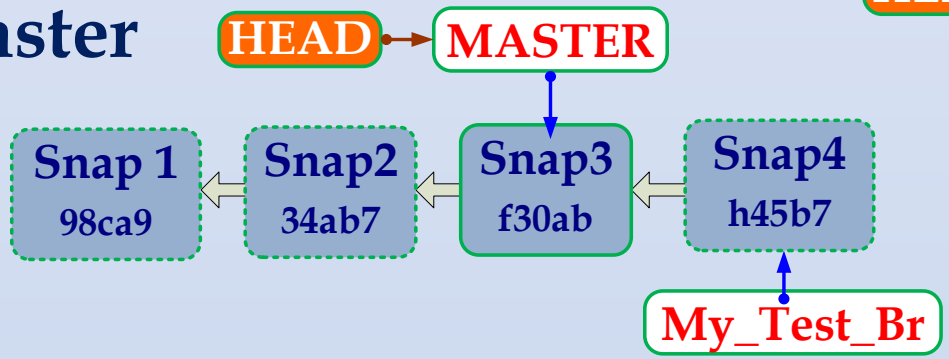
checkout **My_Test_Br пересуває HEAD**

Git. Гілки (branches)

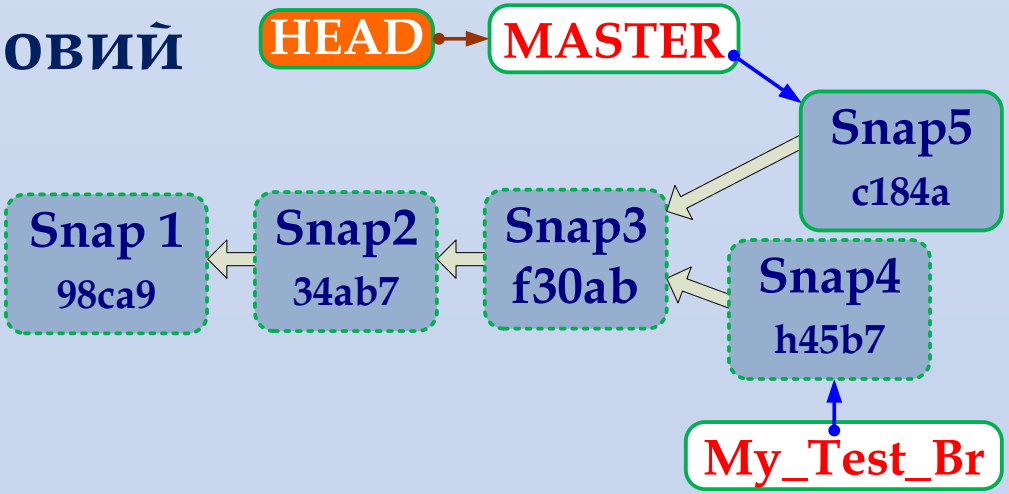
Новий commit - новий snapshot (4)



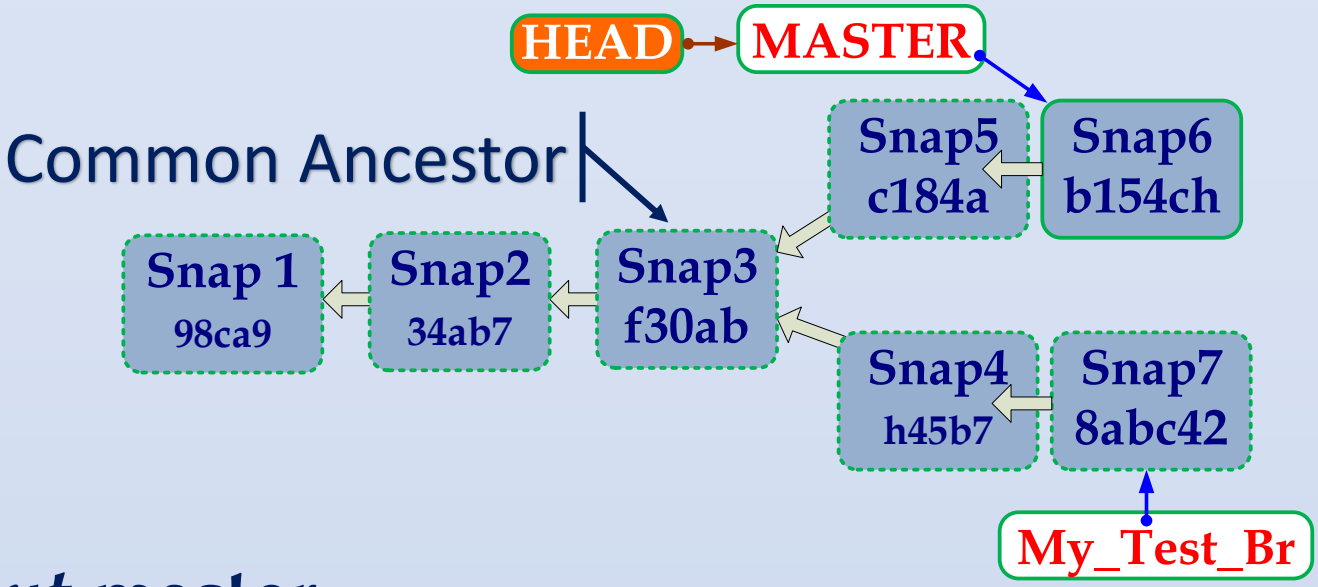
checkout Master



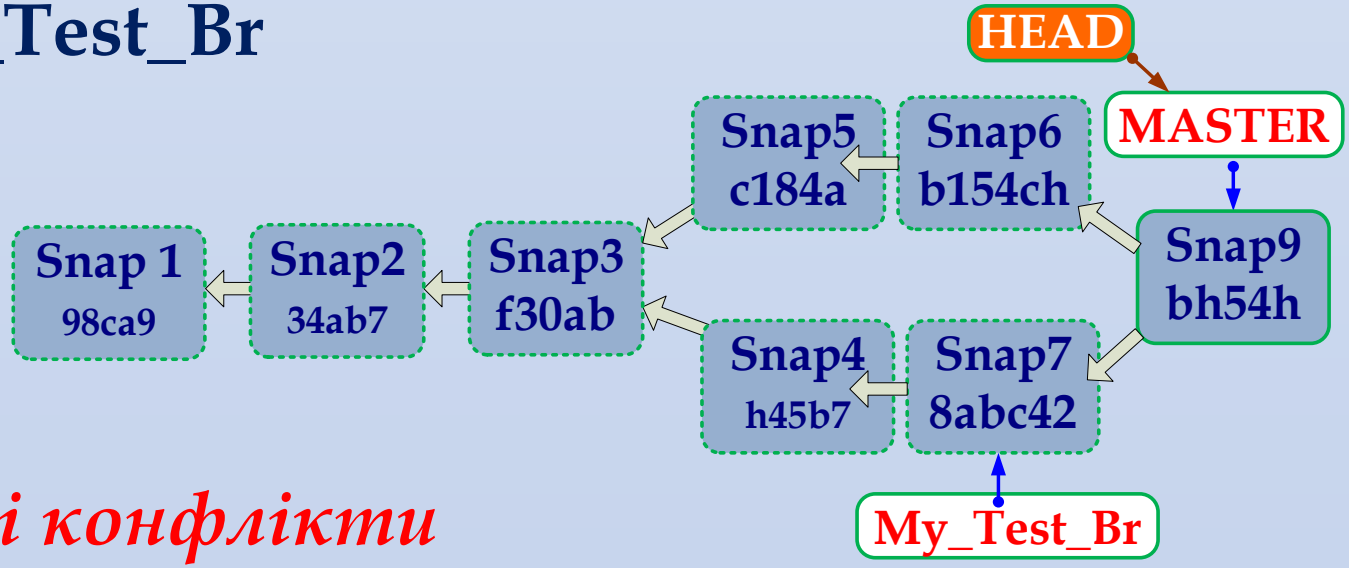
Новий commit - новий snapshot (5)



Git. Гілки, злиття

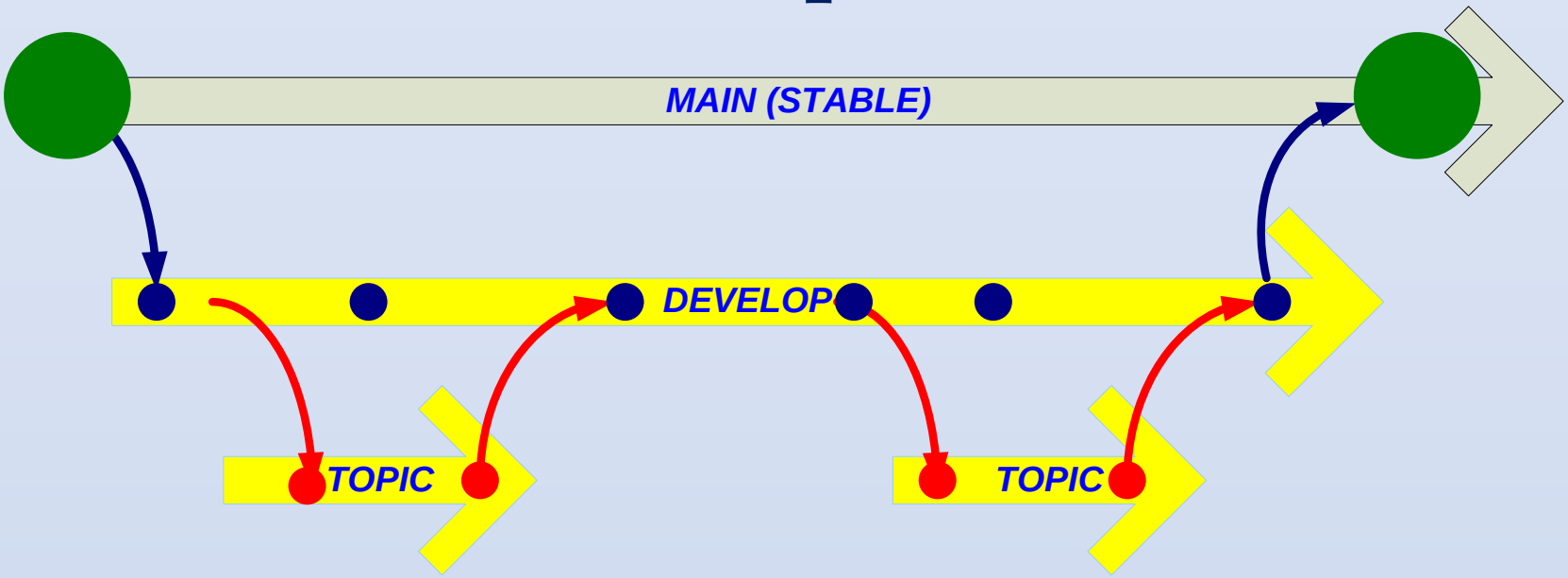


checkout **master**
merge **My_Test_Br**



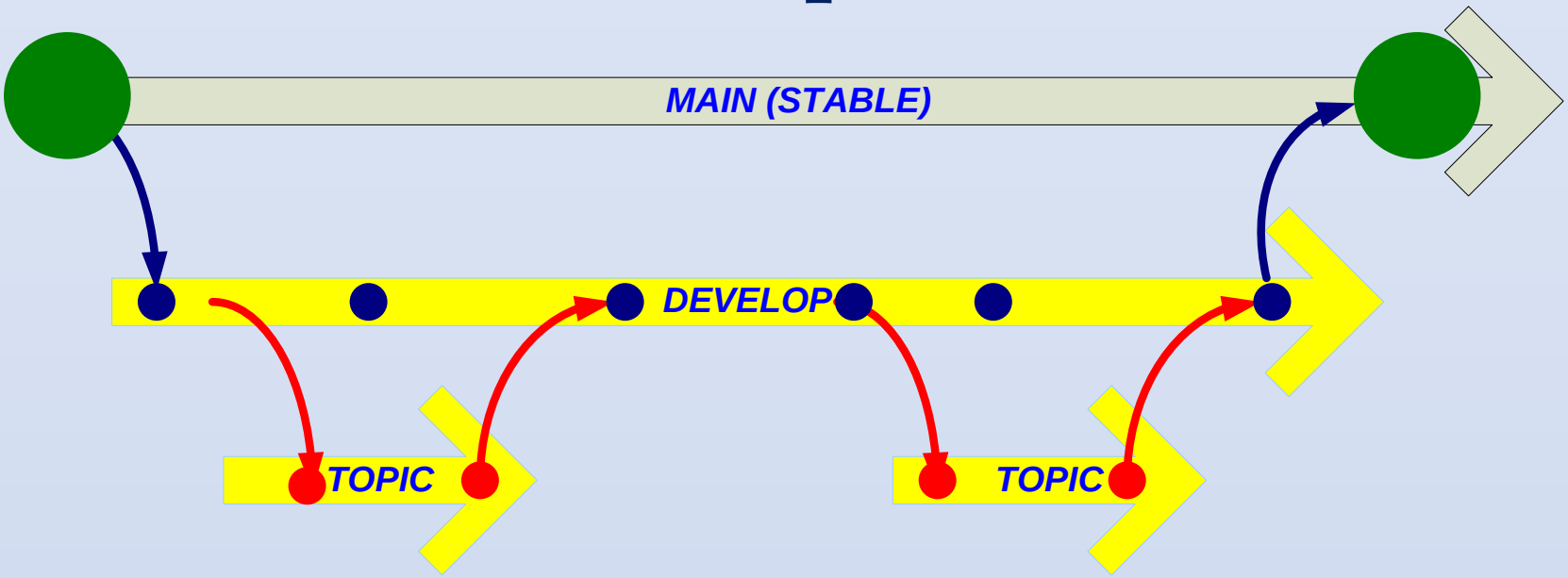
!! Можливі конфлікти

Git. Гілки, тривалість



Гілка MASTER - стабільна версія.
Гілка DEVELOP - гілка тестування стабільності.
Гілка TOPIC - розробник, виправлення .

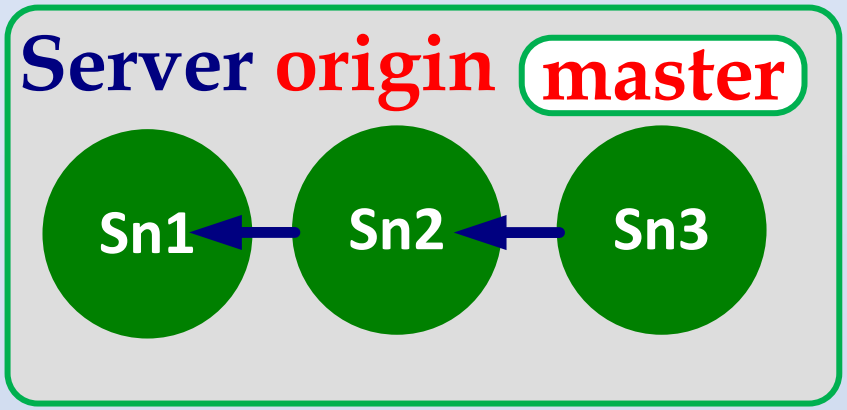
Git. Гілки, тривалість



Гілка MASTER - стабільна версія.
Гілка DEVELOP - гілка тестування стабільності.
Гілка TOPIC - розробник, виправлення .

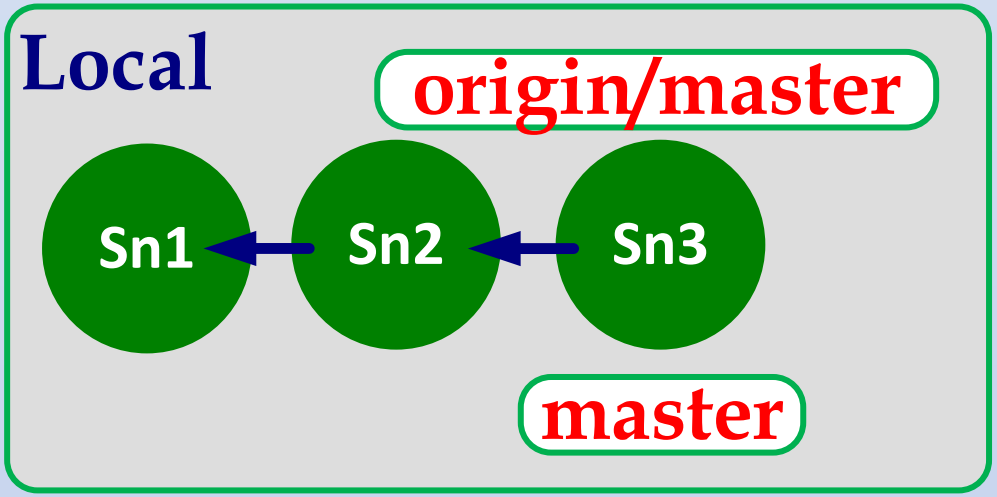
Git. Віддалено-відслідковувані гілки

<відділене_сховище>/<гілка>

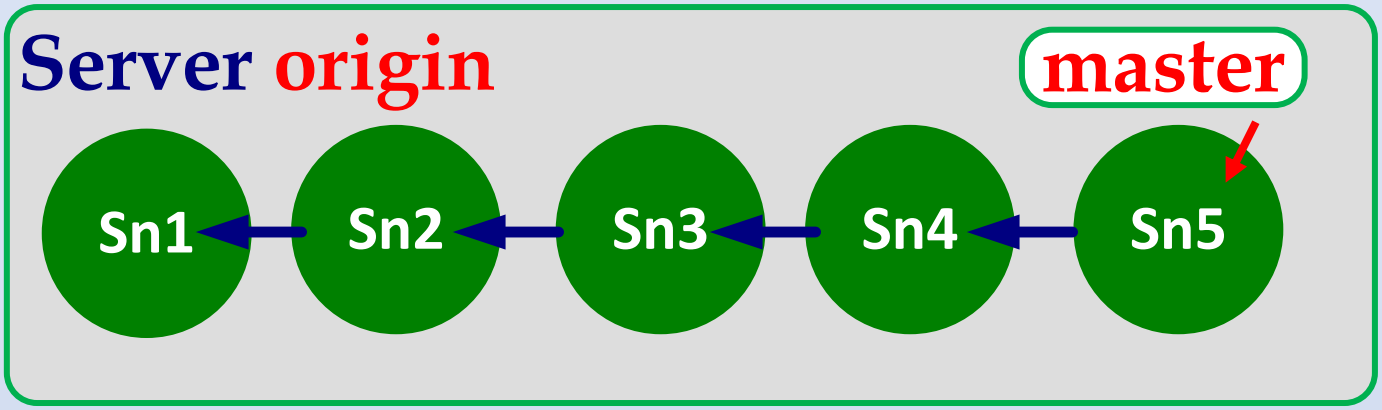


Клонування до локального комп'ютеру

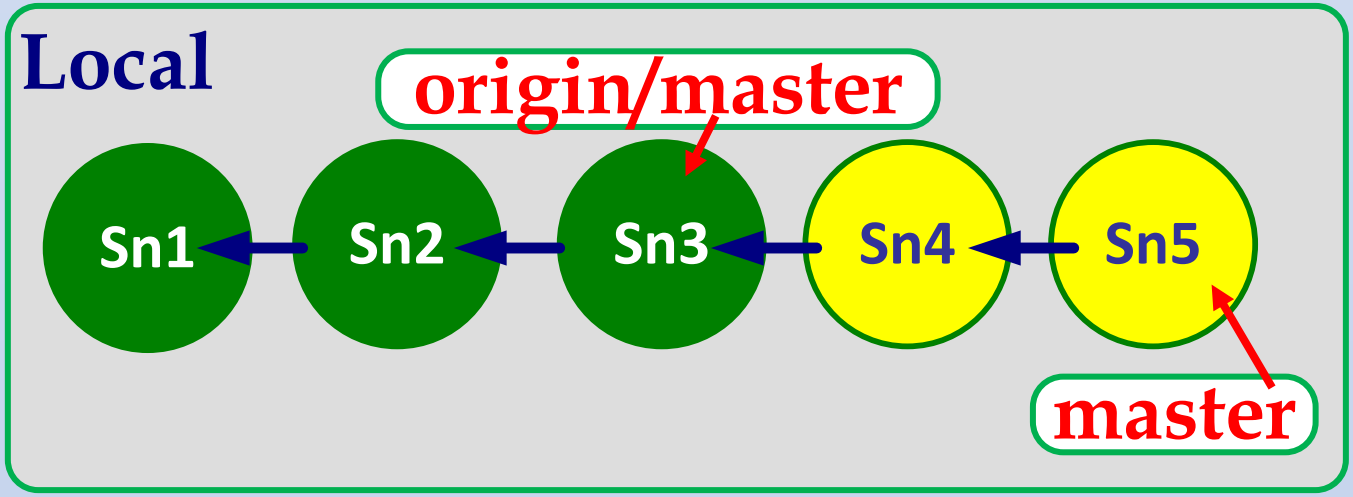
clone <шлях до репозиторію>



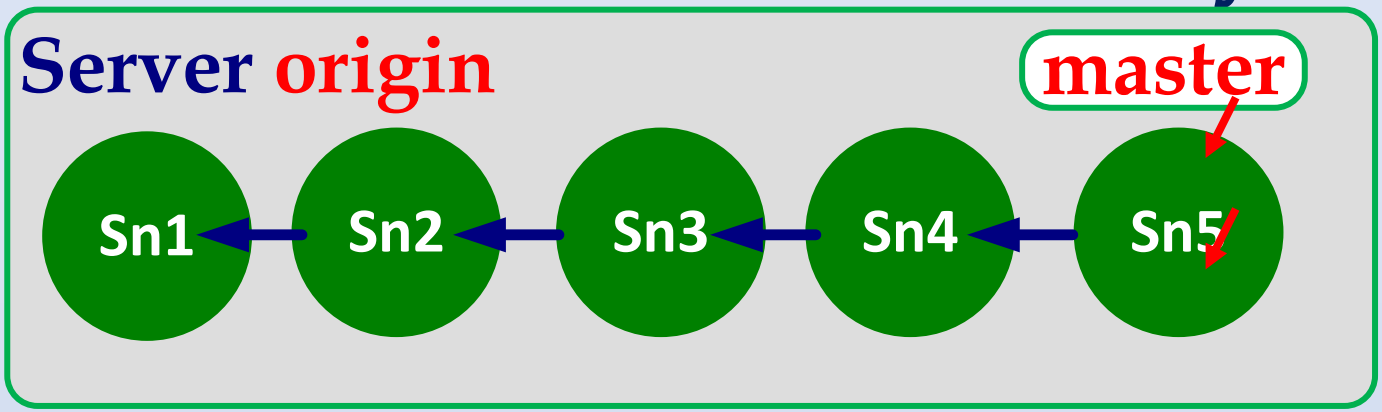
Git. Віддалено-відслідковувані гілки



Різний прогрес – різні історії відділеної гілки та локальної гілки

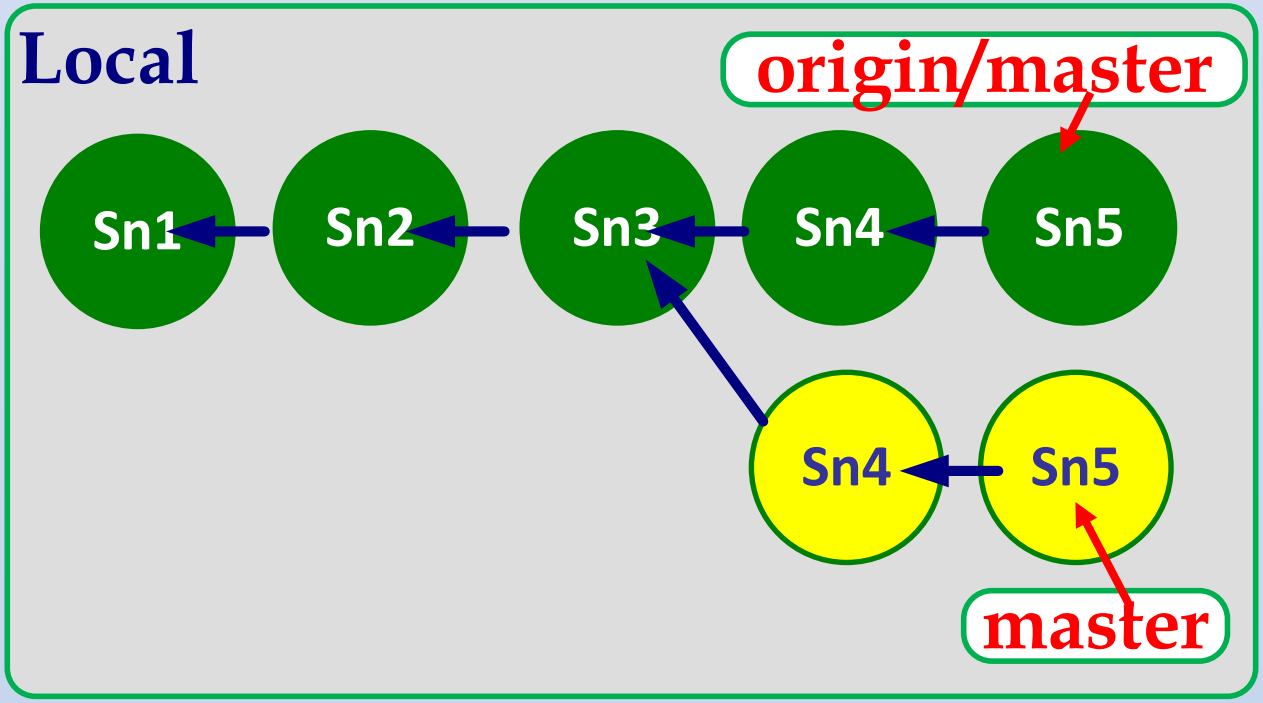


Git. Віддалено-відслідковувані гілки

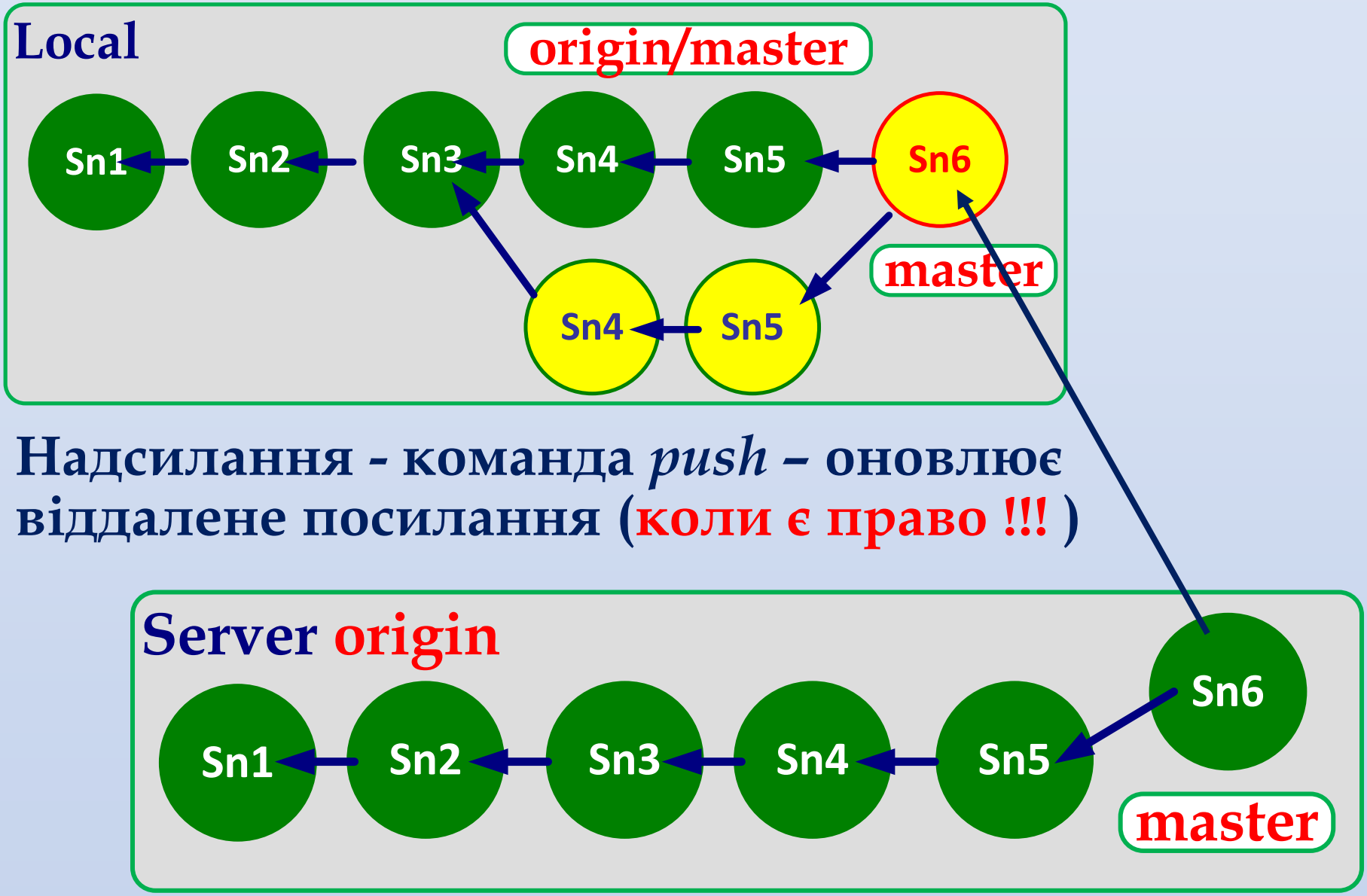


Синхронізація- команда *fetch* – оновлює віддалене посилання

На локальний master не впливає. Зливання - *merge*



Git. Надсилання



Надсилання - команда *push* – оновлює віддалене посилання (коли є право !!!)

Git. Інструментарій

Командний рядок
Git SCM https://gitforwindows.org/

Графічний інтерфейс для Win & macOS
GitHub Desktop - https://desktop.github.com/

Вбудований Git клієнт в Visual Studio

Git хостинг
GitHUB - https://github.com

Сервері рішення Git
WebGitNet - https://github.com/otac0n/WebGitNet

Контрольні питання

- Надайте призначення систем керування версіями та поясніть відмінності локальних, централізованих та децентралізованих СКВ. Визначте переваги та недоліки кожного типу.
- Поясніть принцип зберігання версій в СКВ GIT.
- Визначте стани та процеси контролю файлів в СКВ GIT.
- Поясніть процеси створення та злиття гілок версій в СКВ GIT.
- Опішить роботу з гілками версій, що відслідковуються віддалено.

The END
Лекція 15

ЛЕКЦІЯ 16. ТЕХНОЛОГІЇ #2
Якість & Тестування ПЗ

1. Дефекти ПЗ, визначення, їх класифікація.
2. Якість ПЗ.
3. Тестування, принципи, життєвий цикл дефекту.
4. Рівні, типи та види тестування.
5. Документування.

Проблеми створення ПЗ

ПЗ веде себе невірно (що таке «веде» , що таке «невірно»?).

ЧОМУ? , ЯК ВИПРАВИТИ?

Помилка (*error*) - дія людини, яка породжує неправильний результат.

Дефект, баг (defect, bug) - недолік модуля (компонента) або системи, який може привести до відмови певної функціональності. Дефект, виявлений під час виконання програми, може викликати відмову окремого компонента або всієї системи.

Проблеми створення ПЗ

Результат дефекту: програма може не робити того, що повинна або навпаки - робити те, чого не повинна: відбувається →

Збій (*failure*) - невідповідність фактичного результату (*actual_result*) роботи компонента або системи очікуваного результату (*expected_result*).

Баг існує при одночасному виконанні трьох умов:

- **відомий очікуваний результат;**
- **відомий фактичний результат;**
- **фактичний результат відрізняється від очікуваного результату.**

Не всі баги є причина збоїв - деякі можуть ніяк себе не проявляти і залишатися непоміченими.
Причиною збоїв можуть бути не тільки дефекти, але також і умови навколишнього середовища.

Проблеми створення ПЗ

Причини дефектів

Недолік або відсутність спілкування.

! Замовник розуміє, яким він хоче бачити готовий продукт, але належним чином не пояснив його ідею розробникам і тестувальникам. Вимоги повинні бути доступні і зрозумілі всім учасникам процесу розробки ПЗ.

Складність програмного забезпечення.

Продукт складається з безлічі компонентів, які об'єднуються в складні програмні системи. Все складніше написання і підтримка. Чим складніше // важче робота, тим більше помилок може допустити виконуючий її чоловік.

Причини дефектів

Зміни вимог. Незначні зміни вимог на пізніх етапах розробки вимагають великого обсягу робіт по внесенню змін до системи. !!! Часта *зміна* вимог в бізнесі - *правило*, ніж виняток, тому безперервне тестування і контроль ризиків в таких умовах - прямий обов'язок розробника (відділ забезпечення якості).

Погано документований код. Складно підтримувати і змінювати погано написаний і слабо документований програмний код.

Засоби розробки ПО. Засоби візуалізації, бібліотеки, компілятори, генератори скриптів та інші допоміжні інструменти розробки - це теж часто погано працюють і слабо документовані програми, які можуть стати джерелом дефектів.

Необхідність тестування

Чим пізніше дефект буде виявлений, тим дорожче обійдеться його виправлення, тим більше зусиль для цього буде потрібно.

Висновок : чим раніше в життєвому циклі програми почнеться тестування, тим більшою мірою можна бути впевненим в якості ПЗ.

Якість ПЗ

Якість – «відповідність вимогам» & «придатність до використання».

Комплекс стандартів ISO 9000 – загальні принципи керування якістю продукції .

Стандарт ISO 90003:2004 – «Software engineering — Guidelines for the application of ISO 9001:2000 to computer software». Якість ПЗ - сукупність характеристик ПЗ, що відносяться до його здатності задовольняти встановлені і передбачувані потреби.

Якість ПЗ за ISO/IEC 25000:2014 – здатність ПЗ при заданих умовах задовольняти встановленим або передбачуваним потребам.

Якість ПЗ

Функціональність (*functionality*) **- визначає здатність ПЗ вирішувати завдання, які відповідають зафіксованим і очікуваним потребам користувача, при заданих умовах використання. Характеристика відповідає за те, що ПЗ працює справно і точно, функціонально сумісно, відповідає стандартам галузі і захищене від несанкціонованого доступу.**

Надійність (*reliability*) **- здатність ПЗ виконувати необхідні завдання в позначених умовах протягом заданого проміжку часу або вказану кількість операцій. Атрибути - завершеність і цілісність всієї системи, здатність самостійно і коректно відновлюватися після збоїв в роботі, відмовостійкість.**

Якість ПЗ

Зручність використання (*usability*) / супроводу (*maintainability*) - можливість легкого розуміння, вивчення, використання і привабливості ПЗ для користувача. Легкість, з якою ПО може аналізуватися, тестуватися, змінюватися для виправлення дефектів,.

Ефективність (*efficiency*) - здатність ПЗ забезпечувати необхідний рівень продуктивності у відповідність з виділеними ресурсами, часом і іншими позначеними умовами.

Мобільність (портативність, *portability*) - характеризує ПО з точки зору легкості його перенесення з одного оточення (*software / hardware*) в інше.

Модель якості ПЗ (ISO 9126-1)

Якість ПЗ

- **1. Функціональність**
  - Відповідність стандартам
  - Функціональна сумісність
  - Функціональна справність
  - Безпечність
  - Точність
- **2. Надійність**
  - Стійкість до відмов
  - Завершенність
  - Відновленність
- **3. Ефективність**
  - **Часова ефективність**
  - **Ефективність використання ресурсів**
- **4. Зручність використання. Простота**
  - Встановлення
  - Вивченння
  - Зрозумілість
- **5. Зручність супроводу**
  - Аналізуємость
  - Стабільність
  - Змінність

Тестування ПЗ

Тестування ПЗ це

- процес дослідження ПО з метою отримання інформації про якість програмного продукту;
- процес перевірки відповідності заявлених до продукту вимог і реально реалізованої функціональності, здійснюваний шляхом спостереження за його роботою в штучно створених ситуаціях і на обмеженому наборі тестів, обраних певним чином;
- оцінка системи з тим, щоб знайти відмінності між тим, який система повинна бути і якою вона є.

Тестування - це одна з технік контролю якості (Quality Control), яка включає **планування**, **складання** тестів, безпосередньо **виконання** тестування і **аналіз** отриманих результатів.

Тестування ПЗ

!!!! Тестування ПЗ включає не тільки власне проведення тестів, але і дії, пов'язані з процесом забезпечення якості:

- **аналіз і планування;**
- **розробку тестових сценаріїв;**
- **оцінку критеріїв закінчення тестування;**
- **написання звітів;**
- **рецензування документації (в тому числі і вихідного коду);**
- **проведення статичного аналізу.**

Загальні принципи тестування

1. **Тестування показує наявність дефектів,** але не дозволяє довести їх відсутність.
2. **Повне тестування неможливо** - неможливо провести вичерпне тестування, за виключенням зовсім примітивних випадків.
3. **Раннє тестування** - починати якомога раніше в життєвому циклі розробки ПЗ, зусилля тестування повинні бути сконцентровані на певних цілях.
4. **Накопичення дефектів** - щільність скупчення дефектів в різних елементах програми може відрізнятися. Принцип Парето:

80% проблем містяться в 20% модулів

Загальні принципи тестування

5. Парадокс пестициду – виконуючи одні і ті ж тести знову і знову - вони знаходять все менше нових помилок. Необхідно періодично вносити зміни в набори тестів, і коригувати їх

6. Тестування залежить від контексту - методологія, техніка і тип тестування повинні прямо залежить від природи самої програми.

Також:

- **тестування має проводитися незалежними фахівцями;**
- **необхідно тестувати як позитивні, так і негативні сценарії;**
- **не допускати змін в програмі в процесі тестування;**
- **вказувати очікуваний результат тестів.**

Класифікація дефектів

З точки зору впливу на працездатність ПЗ

1. Blocker – помилка, яка приводе ПЗ в неробочий стан. Подальша робота - **неможлива**.
2. Critical - приводить деякий ключовий функціонал в неробочий стан. Також - суттєве відхилення від бізнес логіки, **неправильна реалізація** необхідних функцій, втрата призначених для користувача даних і т.д.
3. Major - серйозна помилка, яка свідчить про відхилення від бізнес логіки або **порушує роботу** ПЗ. **Не має критичного впливу** на додаток.
4. Minor - **не порушує** функціонал тестованого ПЗ, але є невідповідності результату (дизайн).
5. Trivial - **не впливає** на функціонал.

Класифікація дефектів

З точки зору пріоритетності виправлення

1. High – баг повинен бути виправлений якомога швидше, тому що він критично впливає на працездатність програми.
2. Medium - дефект повинен бути обов'язково виправлений, але він не робить критичний вплив на роботу програми.
3. Low - помилка повинна бути виправлена, але вона не має критичного впливу на програму і усунення може бути відкладено, в залежності від наявності інших більш пріоритетних дефектів.

Життєвий цикл дефекту

1. New – Тестувальник знайшов баг, дефект занесено в «Bug-tracking» систему.
2. Opened. Project manager аналізує дефект вирішує → наступний стан:
   - Deferred – відкладено. Виправлення цього бага не несе цінності на даному етапі розробки або по іншим, відстрочує його виправлення причин.
   - Rejected - відхилена. Дефект не рахується дефектом або вважатися неактуальним.
   - Duplicate - помилка вже раніше була внесена в «Bug-tracking» систему.

Життєвий цикл дефекту

3. Assigned - призначено. Якщо помилка актуальна і повинна бути виправлена в наступній збірці, відбувається призначення на розробника який повинен виправити помилку.
4. Fixed - виправлено. Розробник заявляє, що усунув баг. Залежно від того, чи виправив розробник дефект отримує наступні стани:
   - Verified - перевірено. Даний статус баг отримує після перевірки тестувальником в наступному релізі чи дійсно розробник виправив дефект.
   - Reopened - повторно відкритий. Якщо баг в новому релізі не виправлено.

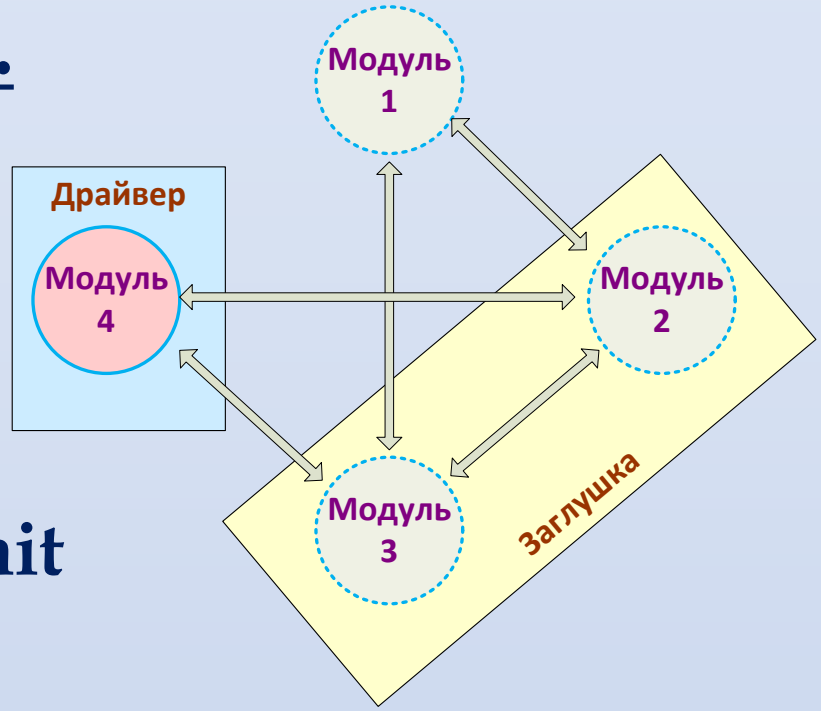
Життєвий цикл дефекту

5. Closed – баг остаточно усунуто.

Рівні тестування.

1. Модульне тестування.

(*Unit testing*) - тестування кожної функціональності програми окремо, в штучно створеному середовищі.



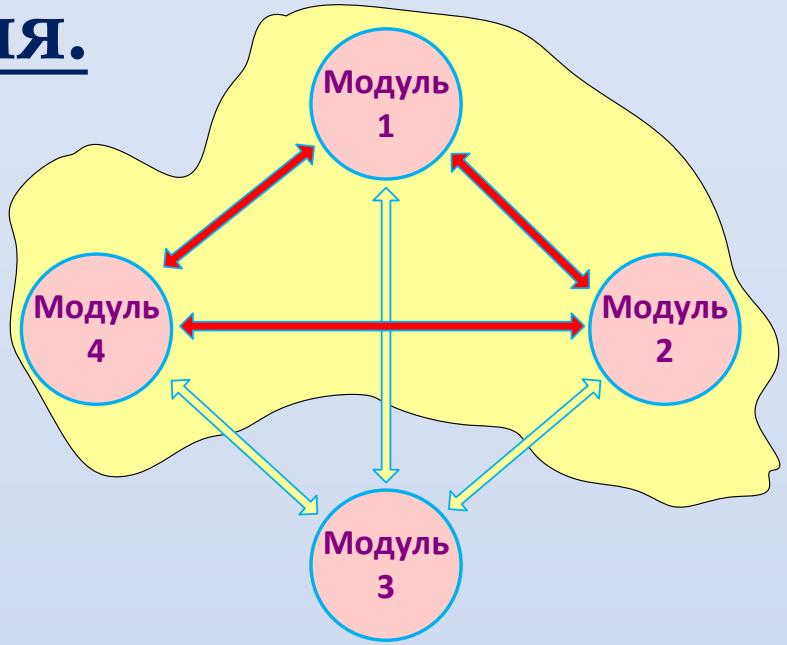
Середовище для деякого unit створюється за допомогою драйверів і заглушок.

Драйвер - певний модуль тесту, який виконують тестований нами елемент.
Заглушка - частина програми, яка симулює обмін даними з тестованим компонентом, виконує імітацію робочої системи.

Рівні тестування.

2. Інтеграційне тестування.

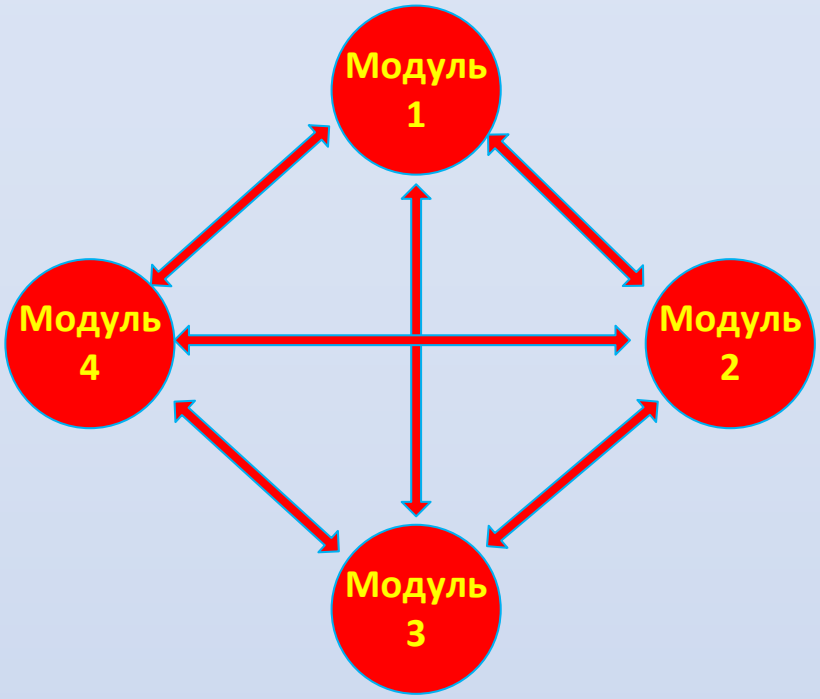
Тестування, при якому на відповідність вимог перевіряється інтеграція модулів, їх взаємодія між собою, а також інтеграція підсистем в одну загальну систему.



Використовуються компоненти, вже перевірені за допомогою модульного тестування, які групуються (множина).
Множина перевіряються відповідно до плану тестування, складеним для них, а об'єднуються вони через свої інтерфейси.

Рівні тестування.

3. Системне тестування.



- тестування ПЗ виконується з метою перевірки відповідності системи вихідним вимогам, як функціональним, так і не функціональним.

Дефекти, що виявляються:

- **Неправильне використання системних ресурсів.**
- **Непередбачені комбінації призначених для користувача даних.**
- **Проблеми з сумісністю оточення.**
- **Непередбачені сценарії використання.**
- **Невідповідність з функціональними вимогами.**
- **Погана зручність використання.**

Рівні тестування.

4. Приймальне тестування.

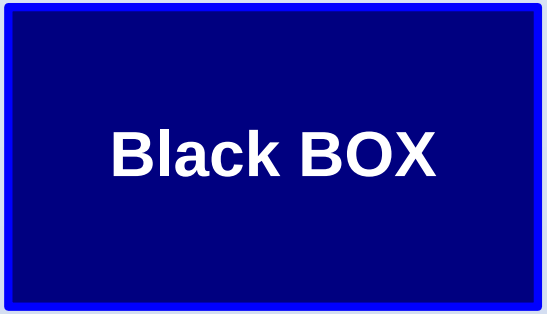
Метою *acceptance* тестування є визначення готовності продукту, що досягається шляхом проходу тестових сценаріїв і випадків, які побудовані на основі специфікації вимог щодо розроблюваного ПЗ.
Проводиться на етапі здачі готового продукту замовнику.
Результатом приймального тестування може стати:

- **Відправлення проекту на доопрацювання.**
- **Ухвалення його замовником, як виконаного завдання.**

Типи тестування

1. White/Black/Grey Box-тестування

Техніка тестування, заснована на роботі виключно з зовнішніми інтерфейсами системи.



Метою тестування технікою Black Box є пошук:
- неправильно реалізовані або відсутні функції;
- помилок інтерфейсу;
- помилок в структурах даних або організації доступу до зовнішніх баз даних;
- помилок поведінки або недостатня продуктивність системи;

Типи тестування

2. White/Black/Grey Box - тестування

Техніка білого (прозорого, відкритого, скляного ящика) тестування передбачає, що внутрішня структура / пристрій / реалізація системи відомі.

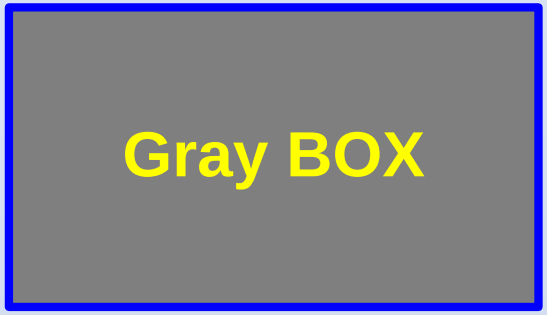
Знання всіх особливостей програми, що тестується і її реалізації - обов'язкові для цієї техніки. Поглиблення під внутрішній устрій системи, за межі її зовнішніх інтерфейсів.

При цьому процедура написання або вибору тест-кейсів на виконується на основі аналізу внутрішнього устрою системи або компонента.

Типи тестування

3. White/Black/Grey Box - тестування

Техніка тестування, заснована на комбінації White Box і Black Box підходів.



Внутрішній устрій програми нам відомо лише частково.
Передбачається доступ до внутрішньої структурі та алгоритмам роботи ПЗ для написання максимально ефективних тест-кейсів, але саме тестування проводиться за допомогою техніки чорного ящика, тобто, з позиції користувача.

Види тестування

1. Функціональне тестування - спрямовано на перевірку відповідності реальних характеристик ПЗ до визначених SRS функціональних вимог. Завдання функціонального тестування – підтвердити, що розроблене ПЗ володіє всім функціоналом, необхідним замовником.

При цьому для тестування створюються тестові випадки (***testcases***), складання яких враховує пріоритетність функцій ПО, які необхідно покрити тестами.
Таким чином можна переконатися в тому, що всі функції розроблюваного продукту працюють коректно при різних типах вхідних даних, їх комбінацій, кількості і т.д.

Види тестування

2. <u>Нефункціональне тестування</u> – тестування властивостей, які не належать до функціональності системи.

- **Надійність -** реакція системи на непередбачені ситуації.
- **Продуктивність -** працездатність системи під різними навантаженнями.
- **Зручність -** дослідження зручності роботи з додатком з точки зору користувача.
- **Масштабованість -** вимоги до горизонтального або вертикального масштабування додатку.
- **Безпека -** захищеність призначених для користувача даних.
- **Мобільність -** переносимість додатку на різні платформи.

Види тестування

3. Регресійне тестування – тестування спрямоване на виявлення так званих регресійних помилок у вже протестованих компонентах ПЗ.

Регресивні помилки - ті ж баги, що з'являються не при написанні програми, а при додаванні в існуючий реліз нових компонентів програми або виправлення інших багів, що може стати причиною виникнення нових дефектів в уже протестованому продукті.

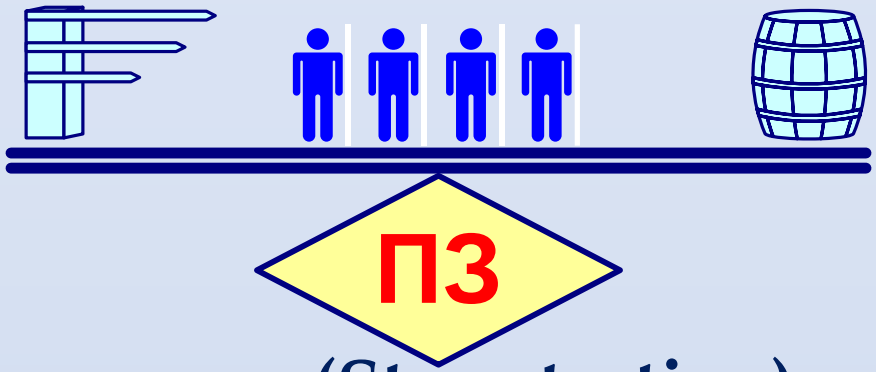
Мета регресійного тестування - переконатися, що виправлення одних багів не спричиняє виникнення інших і що оновлення ПЗ не створила нових дефектів в уже перевіреному коді.

Види тестування

4. Тестування продуктивності – оцінка робочих можливостей, стабільності, обсяг ресурсів, необхідних для використання в умовах різних сценаріїв та навантаження.

- **Тестування навантаження (Loadtesting) - тестування залежності часу відклику для запитів різних типів, з метою підтвердження, що ПЗ працює у відповідність до вимог при звичайному навантаженні.**

Види тестування



- Стрес - тестування (Stresstesting) - тестування робочих можливостей при застосуванні навантаження, що перевищує типові в декількох разів.
- Тестування стабільності або напрацювання на відмову (Stability/Reliabilitytestin) - перевіряє робочу функціональність при тривалій роботі часу, при нормальній програмі навантаження.
- Об’ємне тестування (VolumeTesting) - тестування проводиться із збільшенням кількість використаних даних, які зберігаються і використовуються в додатку..

Документування

1. Test case – перелік, опис та послідовність дій та (або) умов, необхідних для перевірки певної функціональності ПЗ, що розробляється.

Включає:
Action – дія, що виконується.
Expected result очікуваний результат
Test result - фактично отриманий результат.

	A	B	C	D	E	F	G	H	I	J
1	Test Case ID		BU_001	Test Case Description		Test the Login Functionality in Banking				
2	Created By		Mark	Reviewed By		Bill		Version		2.1
3										
4	QA Tester's Log		Review comments from Bill incorprate in version 2.1							
5										
6	Tester's Name		Mark	Date Tested		1-Jan-2017		Test Case (Pass/Fail/Not		Pass
7										
8	S #	Prerequisites:				S #	Test Data			
9	1	Access to Chrome Browser				1	Userid = mg12345			
10	2					2	Pass = df12@434c			
11	3					3				
12	4					4				
13										
14	Test Scenario	Verify on entering valid userid and password, the customer can login								
15										
16	Step #	Step Details		Expected Results		Actual Results			Pass / Fail / Not executed /	
17										
18	1	Navigate to http://demo.guru99.com		Site should open		As Expected			Pass	
19	2	Enter Userid & Password		Credential can be entered		As Expected			Pass	
20	3	Click Submit		Cutomer is logged in		As Expected			Pass	
21	4									
22										
23										

Документування

2. Bug report – технічний документ, який містить в собі повний опис бага, що включає інформацію, як про сам баг (короткий опис – *summary* , серйозність - *severity*, пріоритет - *priority* і т.д.), так і про умовах виникнення даного бага.

Баг репорт повинен містити правильну, єдину термінологію, що описує елементи призначеного для користувача інтерфейсу і події даних елементів, що призводять до виникнення бага.

Short bug description	Severity	Priority	Full bug description
Overflow	High	Medium	If set large values forsides we get wrong area calculation. For example: First side =999; Second side =999; Third side =999 Calculated Area = 15,2

Системи автоматизованого тестування

HP:

- **HP LoadRunner,**
- **HP QuickTest Professional,**
- **HP Quality Center**

IBM:

- **Rational FunctionalTester,**
- **Rational PerformanceTester,**
- **Rational TestStudio**

Segue SilkPerformer
SmartBear Software TestComplete

https://en.wikipedia.org/wiki/Test_automation

Контрольні питання

- Надайте визначення дефектів програмного продукту та головні причини їх появи. Наведіть класифікацію дефектів з точки зору впливу на працездатність ПЗ та пріоритетності. Поясніть необхідність тестування ПЗ.
- Надайте визначення якості програмного продукту, перелік видів якості та пояснить модель якості ПЗ.
- Надайте визначення тестування ПЗ та вкажіть базові принципи тестування. Поясніть життєвий цикл дефекту.
- Надайте перелік рівнів тестування та пояснить їх сутність.
- Поясніть сутність типів тестування White/Black/Gray Box та видів тестування.

The END
Лекція 16

ЛЕКЦІЯ Додаткова.
Розробка та життєвий цикл ПЗ

1. ПЗ: визначення, види.
2. Технології. Вимоги. Стадії.
3. Процеси життєвого циклу ПЗ.
4. Роботи процесу розробки.
5. Каскадна модель життєвого циклу.
6. Ітеративна модель життєвого циклу.
7. Циклічна модель життєвого циклу.

Програмне забезпечення

Програмне забезпечення (ПЗ / Software): комп'ютерні програми, процедури, документація, дані, що забезпечують належне функціонування комп'ютерної системи.

ВИДИ ПЗ (*умовно*):

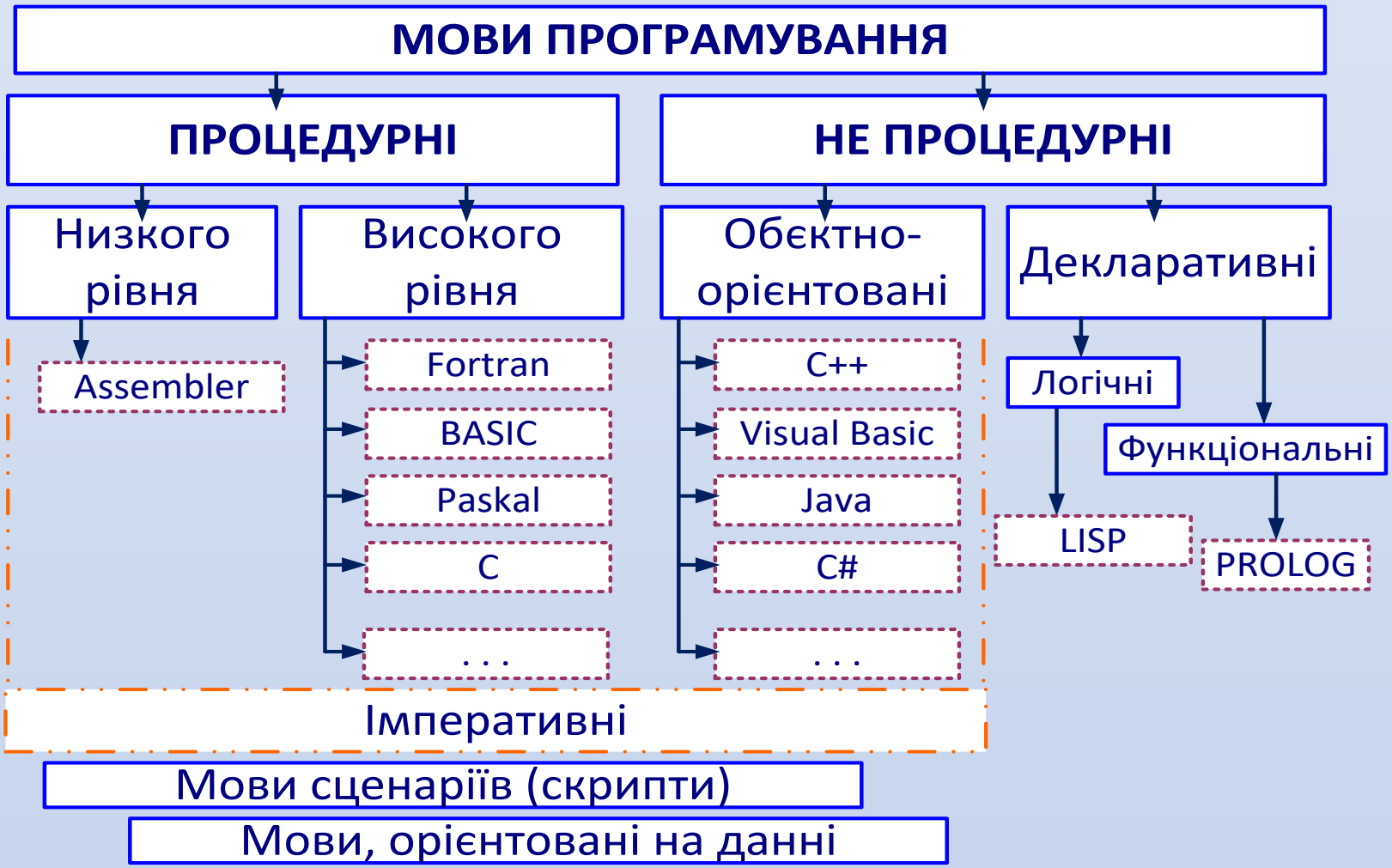
Системне ПЗ (System software).

- **Операційні системи (OS).**
- **Системи програмування.**
- **Сервісні програми (утиліти).**

Прикладне ПЗ (Application software).

Інструментальне ПЗ.

Мови програмування



Розвиток мов (парадигм) програмування

1. Ранні мови програмування (*з 194X років*)

- Операторний підхід

2. Імперативне програмувіання (*з 195X років*)

- Структурний підхід
- Процедурний підхід

3. Декларативне програмувіання (*з 196X років*)

- Функціональне
- Логічне

4.Об'єктно-орієнованне програмування (*з 198X р.*)

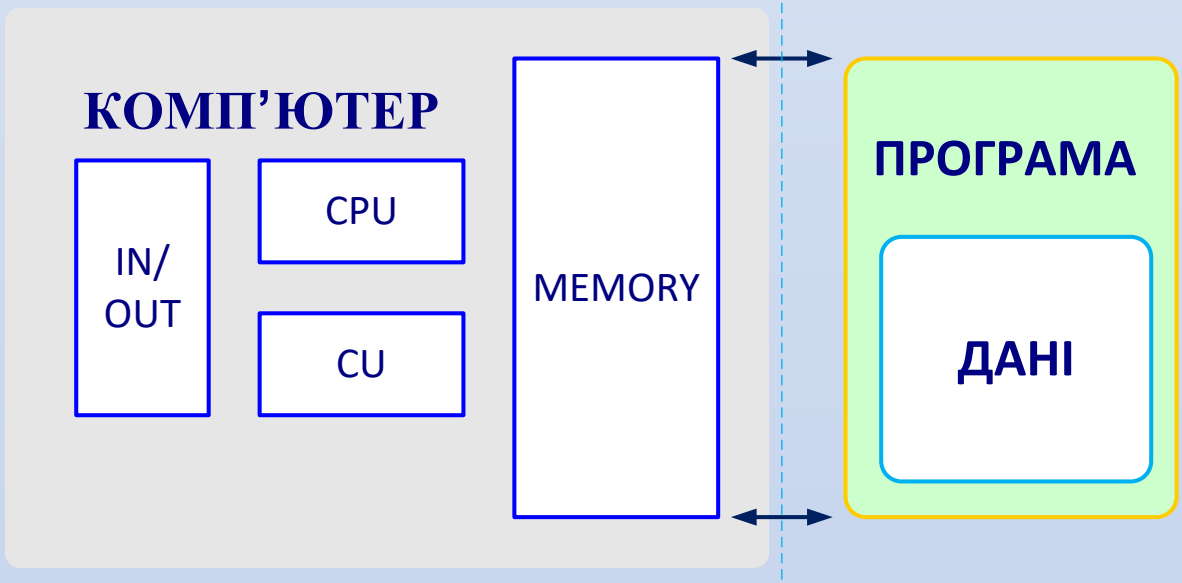
- ООП
- Подієве-кероване
- Паралельне

Компонентне програмування (*з 199X р.*)

Етапи розвитку програмування.

1. Ранні мови - «*стихійне*» програмування

- Технології відсутні!
- Програмування – це мистецтво.
- Один програміст.

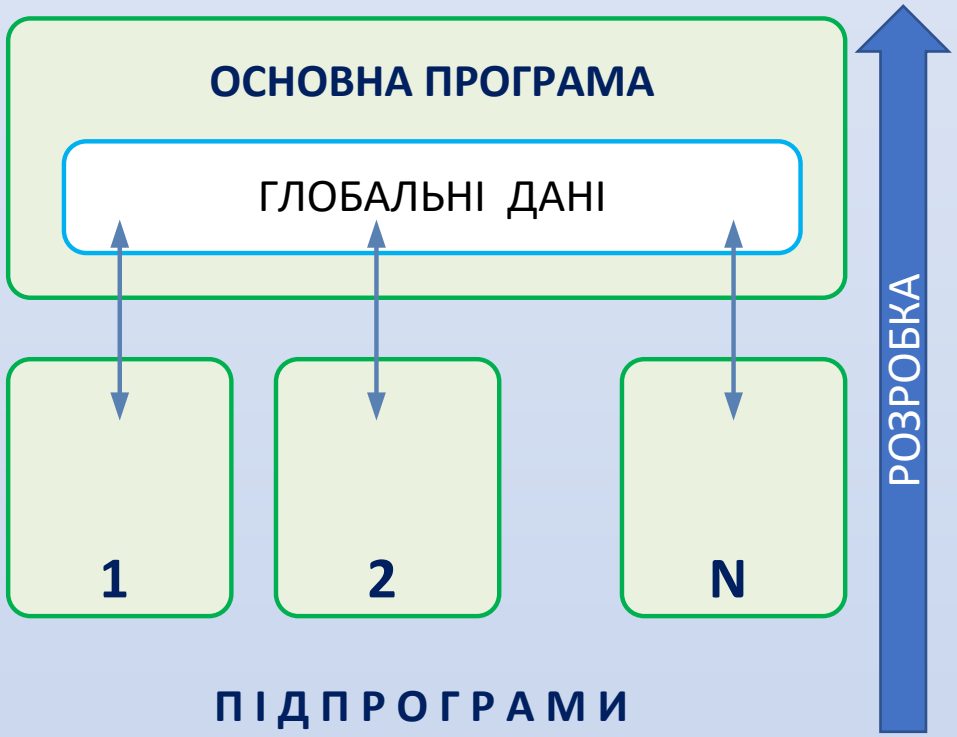


Лінійна послідовність інструкцій (коди, асемблери)
Висока ефективність
Повна залежність від hard!

Етапи розвитку програмування.

1. Ранні мови - підпрограми, бібліотеки

Декілька програмістів : підвищена вірогідність спотворення глобальних даних

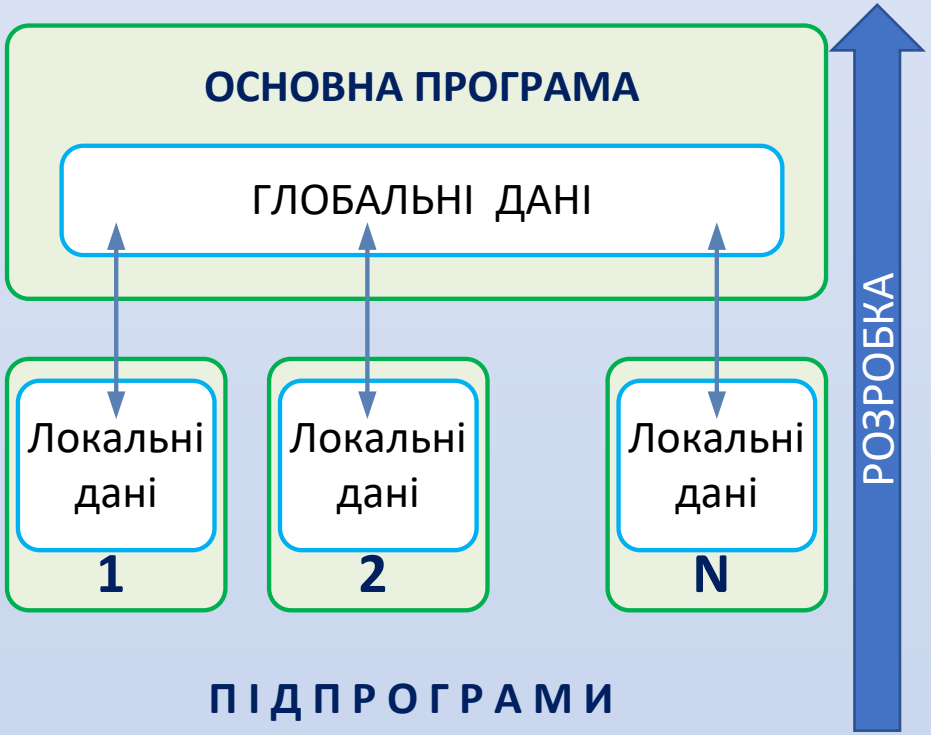


Fortran, Algol, PL/1

Етапи розвитку програмування.

2. Імперативне програмування. Структурний підхід

- **Канонічні структури.**
- **Перші мови «високого рівня».**
- **Абстрагування від конкретного *hard* .**
- **Розгалужені підпрограми.**

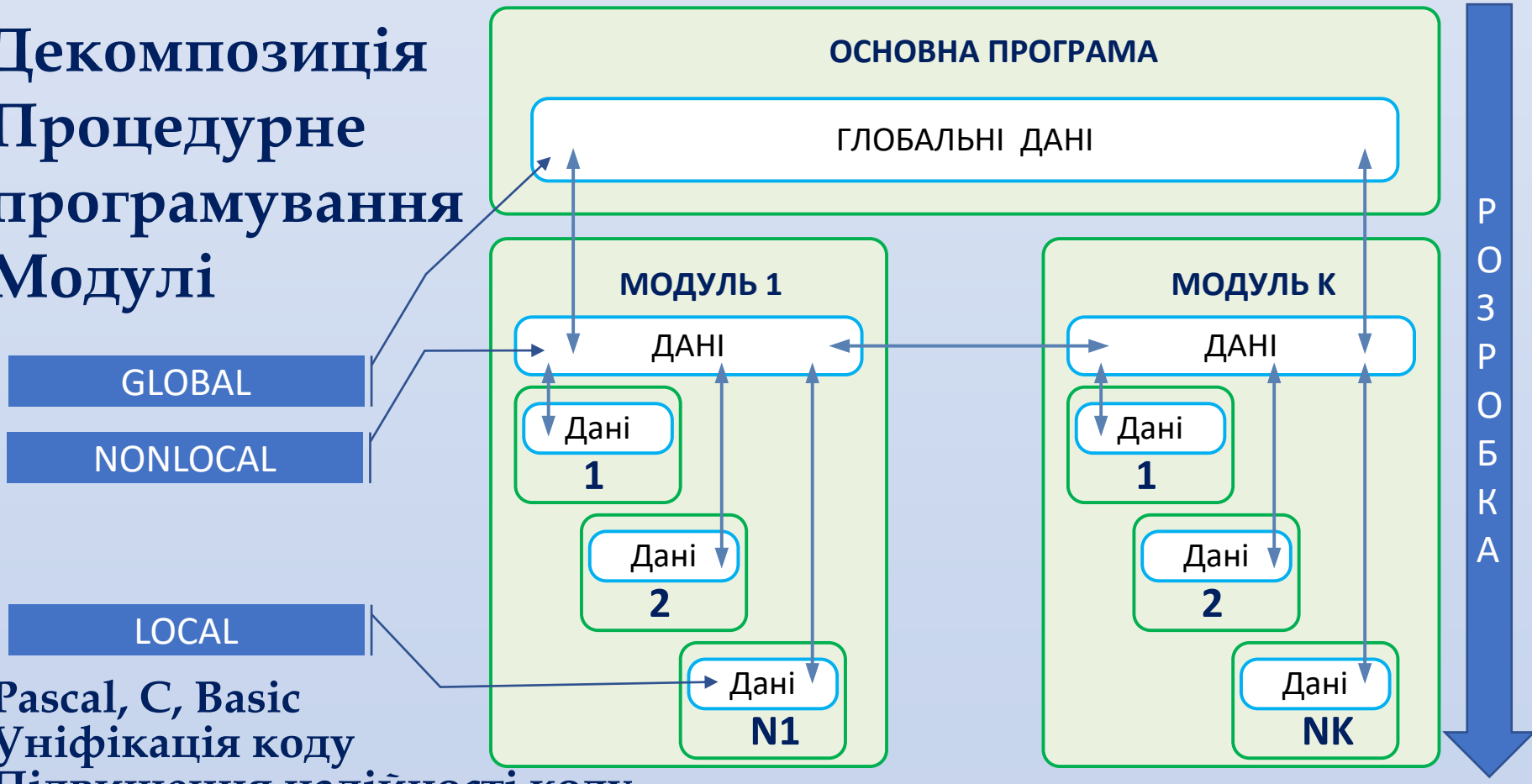


Fortran, Algol, PL/1
Менша залежність від hard
Уніфікація коду

Етапи розвитку програмування.

2. Імперативне програмування. Процедурний підхід

Декомпозиція
Процедурне програмування
Модулі



Pascal, C, Basic
Уніфікація коду
Підвищення надійності коду
Підвищення ефективності праці

Етапи розвитку програмування.

3. Декларативне програмування.

Програма є описом дій, які потрібно виконати, а не набором команд.

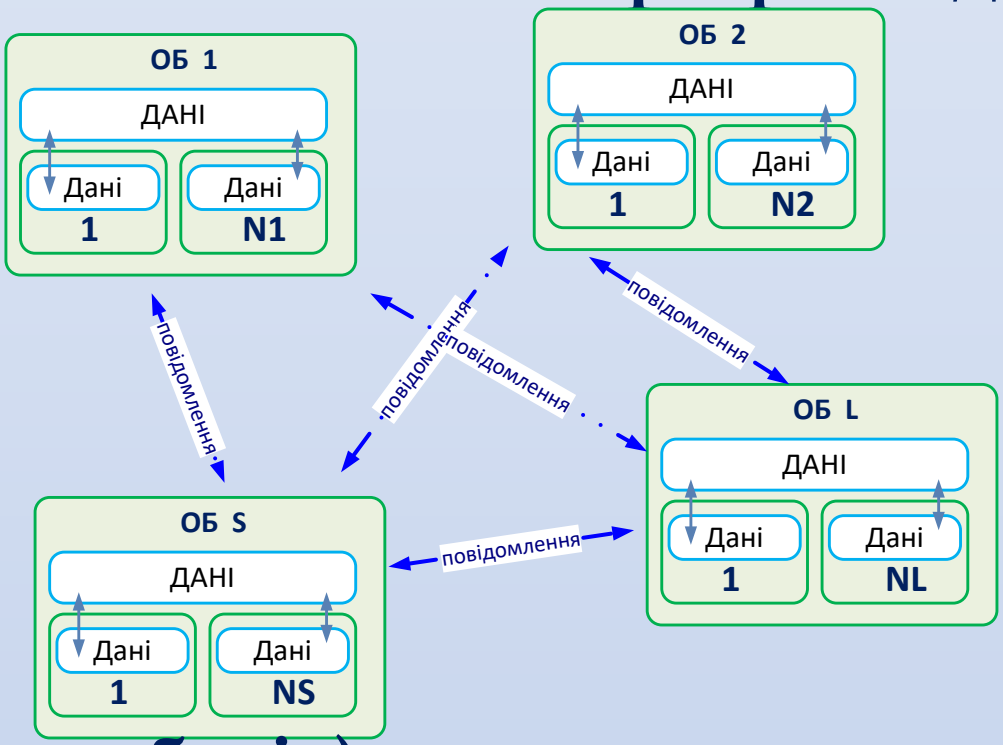
- **Функціональне програмування:** **програма це функція, аргументі якої також функції. Вузька (?) сфера застосування – символьна обробка, математичні перетворення.**
- **Логічне програмування:** **програма є сукупністю логічних висловлювань. Базується на класичній логікі. Вузька сфера застосування – ранні системи прийняття рішень, експертні системи.**

Lisp, Haskel,
Prolog

Етапи розвитку програмування.

4. Об'єктно-орієнтоване програмування

Основна ідея – наблизити текст програми до опису завдання



ОО Програма є описом об'єктів:

- **властивостей (атрибутів),**
- **сукупностей (класів), відносин між ними,**
- **способів їх взаємодії**
- **операцій над об'єктами (методів).**

Етапи розвитку програмування.

4. Об'єктно-орієнтоване програмування

ООП

- **інтуїтивна близькість до довільної предметної області,**
- **моделювання як завгодно складних предметних областей,**
- **орієнтованість на події,**
- **високий рівень абстракції,**
- **повторне використання описів,**
- **параметризація методів обробки об'єктів.**

Event-driven – програма є сукупність можливих сценаріїв (скриптів) обробки даних. Виклик сценарію ініціюється деякою подією (користувач, система).

Етапи розвитку програмування.

5. Компонентне програмування

Компонентне програмування визначає набір правил та обмежень, спрямованих на побудову великих програмних систем, здатних до розвитку впродовж тривалого життєвого циклу.

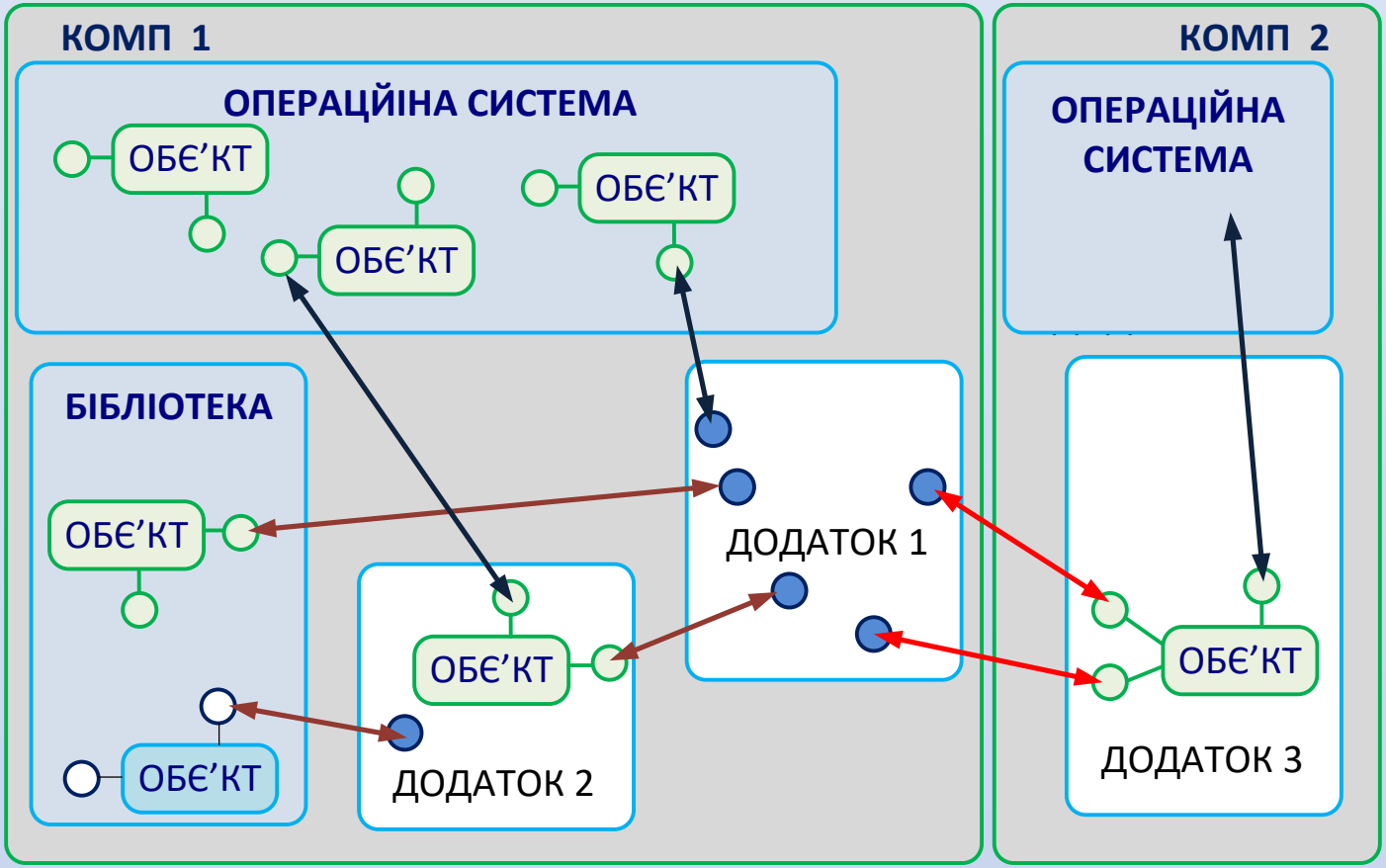
Програмна система складається із окремо створених елементів (компонентів), які викликають один одного через інтерфейси.

Зміни в існуючу систему вносяться додаванням нових компонентів або заміною існуючих.

При цьому нові компоненти, які заміняють раніше створені, повинні наслідувати інтерфейси базового.

Етапи розвитку програмування.

5. Компонентне програмування



CASE технології, шаблони

https://en.wikipedia.org/wiki/Component-based_software_engineering

Етапи розвитку програмування.

5. Компонентне програмування

Програмна ←компонента→ Апаратна

взаємозамінність та надійність

Програмна компонента інкапсулює як структури даних, так і алгоритми, застосовані до структур даних.

Компонентна інженерія програмного забезпечення базується ідеях об'єктно-орієнтованого програмування та їх розвитку:

- **об’єкти ПЗ,**
- **архітектури ПЗ,**
- **рамки (каркаси, frameworks) ПЗ,**
- **шаблони (patterns) ПЗ.**

Етапи розвитку програмування.

5. Компонентне програмування

Object – деяка сутність у цифровому просторі, що володіє певним станом та поведінкою та має визначені властивості (атрибути) та операції над ними (методами).

https://en.wikipedia.org/wiki/Object_

Software architecture – концепція побудови ПЗ: структура програмної системи та методики створення таких структур і систем. Кожна структура містить програмні елементи (об'єкти?!), відносини між ними та властивості як елементів, так і відносин.

https://en.wikipedia.org/wiki/Software_architecture
https://en.wikipedia.org/wiki/Architectural_pattern

Етапи розвитку програмування.

5. Компонентне програмування

Software framework - програмний каркас - це абстракція, в якій ПЗ, що забезпечує загальну функціональність, може вибірково змінюватися додатковим кодом, написаним користувачем, адаптуючи таким чином до потреб додатку. Можуть включати програми підтримки, компілятори, бібліотеки кодів, набори інструментів та інтерфейси прикладного програмування (API), які об'єднують всі різні компоненти, щоб забезпечити розробку проекту чи системи.

https://en.wikipedia.org/wiki/Software_framework

Етапи розвитку програмування.

5. Компонентне програмування

Software design pattern - загальне, багаторазове вирішення проблем, що часто зустрічається в заданому контексті в розробці ПЗ.

!!! Шаблон проектування показує, як вирішити проблему, та який може бути використано у багатьох різних ситуаціях.

Шаблони дизайну - це формалізовані найкращі практики, які програміст може використовувати для вирішення загальних проблем при проектуванні програми чи системи.

https://en.wikipedia.org/wiki/Software_design_pattern

Технологія (програмування)

Технологія (від грец. *τεχνη* — майстерність, техніка + грец. *λογος* — передавати) — наука («корпус знань) про способи (набір і послідовність операцій, їх режими) забезпечення потреб людства за допомогою технічних засобів (знарядь, пристроїв, …).

Технологія - комплекс наукових та інженерних знань, втілених в способах і засобах праці, наборах матеріально-речових факторів виробництва, видах їх поєднання для створення певного продукту або послуги.

Технологія програмування - сукупність методів, процесів і засобів, що застосовуються для створення програмного продукту.

Технологія (програмування)

Компоненти технологічного процесу (*програмування*) :

- **мета реалізації процесу;**
- **об'єкт (предмет), що підлягає технологічним змінам;**
- **способи і методи впливу на об'єкт ;**
- **засоби технологічного впливу на об'єкт ;**
- **впорядкованість і організація, які протиставлені стихійним процесам.**

Технологія (програмування)

Загальні вимоги до технологічного процесу (*програмування*) :

- **високий ступінь поділу процесу на стадії (фази);**
- **системна повнота (цілісність) процесу, який повинен включати весь набір елементів, що забезпечують необхідну завершеність дій в досягненні поставленої мети;**
- **регулярність процесу і однозначність його фаз, їх стандартизація і уніфікація;**
- **технологія є нерозривно пов'язаною із процесом - сукупністю та послідовністю дій, які виконуються в часі;**
- **автоматизація операцій на всіх стадіях.**

Технологія програмування

Технологія програмування

Особливості сучасних систем ПЗ

- Складність опису - безліч функцій, процесів, даних, складні взаємозв'язки.
- Безліч тісно взаємодіючих компонентів - мають свої локальні завдання і цілі функціонування.
- Необхідність інтеграції існуючих і нових додатків.
- Функціонування в неоднорідному середовищі, на декількох апаратних платформах, ОС.
- Істотна тимчасова протяжність створення та існування ПЗ.
- Колективна розробка + різнорідність і роз'єднаність окремих груп розробників.

Технологія програмування

Базові стадії (фази) технологічного процесу створення програмного продукту

- Аналіз.
- Проектування.
- Програмування.
- Тестування + налагодження.
- Впровадження.
- Експлуатація + супровід.

ISO/IEC 12207

Група стандартів ISO/IEC 12207: «System and software engineering — Software life cycle processes».

ДСТУ ISO/IEC 12207:2016 «Інженерія систем і програмного забезпечення. Процеси життєвого циклу програмного забезпечення»

http://online.budstandart.com/ua/catalog/doc-page.html?id_doc=71824

ISO/IEC 12207. Термінологія

Система (system): комплекс, що складається з процесів, технічних і програмних засобів, пристроїв і персоналу, що володіє можливістю задовольняти встановленим потребам або цілям.

Програмно-апаратний засіб (firmware): поєднання технічних пристроїв і машинних команд або використовуваних обчислювальною машиною даних, що зберігаються на технічному пристрої у вигляді постійного програмного засобу.

ISO/IEC 12207. Термінологія

Програмний продукт (software product): набір машинних програм, процедур і, можливо, пов'язаних з ними документації і даних.

Програмна послуга (software service): виконання робіт, завдань чи обов'язків, пов'язаних з програмним продуктом (розробка, супровід або експлуатація).

Програмний модуль (software unit): Окремо компілюємо частина програмного коду (програми).

ISO/IEC 12207. Термінологія

Замовлення (acquisition): процес придбання системи, програмного продукту або програмної послуги.

Замовник (acquirer): організація, яка набуває або отримує систему, програмний продукт або програмну послугу від постачальника.

Постачальник (supplier): організація, яка укладає договір із замовником на поставку системи, програмного продукту або програмної послуги.

Користувач (user) : особа чи організація, яка використовує діючу систему для виконання конкретної функції (Користувач →інші ролі: замовник, розробник, супроводжувач).

ISO/IEC 12207. Термінологія

Технічне завдання (statement of work): документ, який використовується замовником в як засіб для опису і визначення завдань, які виконуються при реалізації договору.

Випуск (release): конкретна версія елемента конфігурації, яка доступна для реалізації конкретної мети (наприклад, тестований випуск).

Версія (version): певний екземпляр об’єкту.

ISO/IEC 12207. Термінологія

Модель життєвого циклу (life cycle model): структура, що складається з процесів, робіт і завдань, що включають в себе розробку, експлуатацію та супровід програмного продукту, що охоплює життя системи від встановлення вимог до неї до припинення її використання.

Процес (process): набір взаємопов'язаних робіт, які перетворять вихідні дані у вихідні результати.

ISO/IEC 12207. Процеси

1. Основні процеси

- 1.1. Замовлення
- 1.2. Поставка
- 1.3. Розробка
- 1.4. Експлуатація
- 1.5. Супровід

2. Допоміжні процеси

- 2.1. Докуметування
- 2.2. Управління конфігурацією
- 2.3. Забезречення якості
- 2.4. Веріфікація
- 2.5. Атестація
- 2.6. Сумісний аналіз
- 2.7. Аудит
- 2.8. Рішення проблем

3. Організаційні процеси

- 3.1. Керування
- 3.2. Створення інфраструктури
- 3.3. Вдосконалення
- 3.4. Навчання

ISO/IEC 12207. Основні процеси

Процеси, які реалізуються під управлінням основних сторін (замовник, постачальник, розробник, оператор, персонал супроводу), залучених в життєвий цикл програмних засобів.

1.1. Замовлення: роботи замовника - організації, яка набуває систему, програмний продукт або програмну послугу.

1.2. Поставка: роботи постачальника, тобто організації, яка поставляє систему, програмний продукт або програмну послугу замовнику.

ISO/IEC 12207. Основні процеси

1.3. Розробка: роботи розробника, тобто організації, яка проектує і розробляє програмний продукт.

1.4. Експлуатація: роботи оператора, тобто організації, яка забезпечує експлуатаційне обслуговування системи в заданих умовах в інтересах користувачів.

1.5. Супровід: роботи персоналу супроводу, тобто організації, яка надає послуги з супроводу програмного продукту (контрольовані зміні програмного продукту з метою збереження його початкового стану і функціональних можливостей.

ISO/IEC 12207. Допоміжні процеси

Допоміжний процес є цілеспрямованою складовою частиною основного процесу, забезпечує успішну реалізацію та якість виконання програмного проекту.

2.1. Документування: роботи по опису інформації, яка видається в процесі життєвого циклу.

2.2. Управління конфігурацією: роботи по управлінню конфігурацією системи.

2.3. Забезпечення якості: роботи за об'єктивним забезпечення того, щоб програмні продукти і процеси відповідали вимогам, встановленим для них, і реалізовувалися в рамках затверджених планів.

ISO/IEC 12207. Допоміжні процеси

2.4. Верифікації: роботи (замовника, постачальника або незалежної сторони) по верифікації програмних продуктів у міру реалізації програмного проекту.

2.5. Атестація: роботи (замовника, постачальника або незалежної сторони) по атестації програмних продуктів програмного проекту.

2.6. Спільний аналіз: роботи з оцінки стану і результатів будь-якої роботи. Даний процес може використовуватися двома будь-якими сторонами, коли одна зі сторін (що перевіряє) перевіряє іншу сторону (перевіряється) на спільній нараді.

ISO/IEC 12207. Допоміжні процеси

2.7. Аудит: роботи по визначенню відповідності вимогам, планам і договорам. Даний процес може використовуватися двома сторонами, коли одна зі сторін (що перевіряє) контролює програмні продукти або роботи іншого боку (перевіряється).

2.8. Вирішення проблем: процес аналізу та усунення проблем (включаючи невідповідності), незалежно від їх характеру і джерела, які були виявлені під час здійснення розробки, експлуатації, супроводу або інших процесів.

ISO/IEC 12207. Організаційні процеси

Процеси створення і реалізації основної структури, які охоплюють взаємопов'язані процеси життєвого циклу і відповідний персонал, а також для постійного вдосконалення даної структури і процесів.

3.1. Управління: Визначає основні роботи з управління, включаючи управління проектом, при реалізації процесів життєвого циклу.

3.2. Створення інфраструктури: роботи по створення основної структури процесу життєвого циклу.

ISO/IEC 12207. Організаційні процеси

3.3. Удосконалення: основні роботи, які організація (замовника, постачальника, розробника, оператора, персоналу супроводу або адміністратора іншого процесу) виконує при створенні, оцінці, контролі і удосконалення обраних процесів життєвого циклу.

3.4. Навчання: роботи за відповідним навчанням персоналу.

ISO/IEC 12207. 1.3. Розробка

1.3. Розробка

- 1.3.1. Підготовка процесу розробки
- 1.3.2. Аналіз вимог до системи
- 1.3.3. Проектування системної архітектури
- 1.3.4. Аналіз вимог до програмних засобів
- 1.3.5. Проектування програмної архітектури
- 1.3.6. Технічне проектування програмних засобів
- 1.3.7. Програмування та тестування програмних засобів
- 1.3.8. Збірка програмних засобів
- 1.3.9. Випробування програмних засобів
- 1.3.10. Збірка системи
- 1.3.11. Випробування системи
- 1.3.12. Забезпечення прийомки програмних засобів

ISO/IEC 12207. 1.3. Розробка

1.3.1. Підготовка процесу розробки:
розробник повинен:

- **визначити чи вибрати модель життєвого циклу програмних засобів, відповідну області реалізації, величини та складності проекту. Повинні бути вибрані і структуровані в моделі життєвого циклу програмних засобів роботи і завдання процесу розробки. Повинен бути розроблено план проведення робіт.**

- **!!!Документальне оформлення (2.1.)**

ISO/IEC 12207. 1.3. Розробка

1.3.2. Аналіз вимог до системи.

Розробник повинен

- **виконати аналіз сфери застосування розроблюваної системи з точки зору визначення вимог до неї. Технічні вимоги до системи повинні охоплювати:**
  - **функції і можливості системи;**
  - **комерційні та організаційні вимоги;**
  - **вимоги користувача;**
  - **вимоги безпеки і захисту;**
  - **ергономічні вимоги;**
  - **вимоги до інтерфейсів;**
  - **експлуатаційні вимоги;**
  - **вимоги до супроводу;**
  - **проектні обмеження та кваліфікаційні вимоги.**
- **Технічні вимоги до системи повинні бути документально оформлені.**

ISO/IEC 12207. 1.3. Розробка

1.3.3. Проектування системної архітектури.

Розробник повинен:

- **Визначити загальну архітектура системи (архітектура верхнього рівня).**
- **Вказати об'єкти технічних і програмних засобів і ручних операцій.**
- **Визначити та забезпечити розподіл всіх вимог до системи між об'єктами архітектури.**
- **Визначити об'єкти конфігурації технічних і програмних засобів і ручних операцій на основі об'єктів архітектури.**
- **Документально оформити прив’язку системної архітектури та вимог до системи щодо встановлених об'єктів.**

ISO/IEC 12207. 1.3. Розробка

1.3.4. Аналіз вимог до програмних засобів :

Розробник повинен:

- **Встановити і документально оформити такі вимоги до програмних засобів, включаючи технічні вимоги до характеристик якості :**
  - **функціональні та технічні вимоги, включаючи продуктивність, фізичні характеристики і навколишні умови, під які повинен бути створений програмний об'єкт архітектури (далі - програмний об'єкт);**
  - **вимоги до зовнішніх інтерфейсів програмного об'єкта архітектури;**
  - **кваліфікаційні вимоги;**
  - **вимоги захисту, включаючи вимоги, що відносяться до допустимої точності інформації;**

ISO/IEC 12207. 1.3. Розробка

1.3.4. Аналіз вимог до програмних засобів :

- **вимоги безпеки, включаючи вимоги, що стосуються способів експлуатації та супроводу, впливу навколишнього середовища і травмобезпечності персоналу;**
- **ергономічні вимоги, включаючи вимоги, що відносяться до ручних операцій, взаємодії "людина-машина", персоналу і областям, які вимагають концентрації уваги людини, пов'язаних з чутливістю об'єкта до помилок людини і навченості персоналу;**
- **вимоги до визначення даних і бази даних;**
- **вимоги щодо введення в дію та приймання поставленого програмного продукту на об'єктах експлуатації та супроводу;**
- **вимоги до документації користувача;**
- **вимоги до експлуатації об'єкта користувачем;**
- **вимоги до обслуговування користувача.**

ISO/IEC 12207. 1.3. Розробка

1.3.4. Аналіз вимог до програмних засобів :

- **Оцінити вимоги до програмних засобів за такими критеріям (при цьому результати оцінок повинні бути документально оформлені):**
  - **врахування вимог до системи і проекту системи;**
  - **зовнішня узгодженість з вимогами до системи;**
  - **внутрішня узгодженість вимог до об'єктів між собою;**
  - **можливість тестування вимог;**
  - **здійсненність програмного проекту;**
  - **можливість експлуатації і супроводу.**

ISO/IEC 12207. 1.3. Розробка

1.3.5. Проектування програмної архітектури:
Розробник повинен:

- **Трансформувати вимоги до програмного об'єкту в архітектуру, яка описує загальну структуру об'єкта і визначає компоненти вимог до програмному об'єкту.**
- **Забезпечити розподіл всіх вимог до програмному об'єкту між його компонентами і подальше їх уточнення з точки зору полегшення технічного проектування. Архітектура програмного об'єкта повинна бути документально оформлена.**
- **Розробити і документально оформити загальні (ескізні) проекти зовнішніх інтерфейсів програмного об'єкта, інтерфейсів між компонентами та баз даних.**

ISO/IEC 12207. 1.3. Розробка

1.3.5. Проектування програмної архітектури:

- **Розробити і документально оформити попередні версії документації користувача.**
- **Визначити попередні загальні вимоги до випробувань (програмного об'єкта і графік збірки програмного продукту.**
- **Оцінити архітектуру програмного об'єкта і ескізні проекти:**
  - **врахування вимог до програмного об'єкту;**
  - **зовнішня узгодженість з вимогами до програмного об'єкту;**
  - **внутрішня узгодженість між компонентами програмного об'єкта;**
  - **відповідність методів проектування і використовуваних стандартів;**
  - **e) можливість технічного проектування;**
  - **f) можливість експлуатації і супроводу.**

ISO/IEC 12207. 1.3. Розробка

1.3.6. Технічне проектування програмних засобів.

Розробник повинен:

- **Розробити технічний проект для кожного компонента програмного об'єкта.**
  - **Компоненти програмного об'єкта повинні бути уточнені на рівні програмних модулів, які можна програмувати (кодувати), компілювати і тестувати незалежно. Повинно бути забезпечено розподіл технічних вимог до компонентів програмного об'єкта між програмними модулями.**
- **Розробити і документально оформити технічні проекти**
  - **зовнішніх інтерфейсів програмного об'єкта:**
  - **інтерфейсів між компонентами програмного об'єкта і між програмними модулями, баз даних;**
  - **технічні проекти інтерфейсів повинені забезпечити виконання програмування без потреби у додатковій інформації.**

ISO/IEC 12207. 1.3. Розробка

1.3.7. Програмування та тестування програмних засобів. Розробник повинен:

- **Розробити і документально оформити такі продукти:**
  - **кожен програмний модуль і базу даних;**
  - **процедури випробувань (тестування) та дані для тестування кожного програмного модуля і бази даних.**
- **Протестувати кожен програмний модуль і базу даних, гарантуючи, що вони відповідають встановленим вимогам. Результати тестування повинні бути документально оформлені.**
- **Уточнити документацію користувача (при необхідності).**
- **Уточнити загальні вимоги до тестування і програму збірки програмних засобів.**

ISO/IEC 12207. 1.3. Розробка

1.3.7. Програмування та тестування програмних засобів. Розробник повинен:

- **Оцінити запрограмовані елементи програмного об'єкта і результати їх тестування за наступними критеріями (при цьому результати оцінок повинні бути документально оформлені):**
  - **врахування вимог до програмного об'єкту і проекту об'єкта в цілому;**
  - **зовнішнє відповідність вимогам і проекту програмного об'єкта;**
  - **внутрішнє відповідність між вимогами до програмних модулів;**
  - **тестове покриття всіх модулів;**
  - **відповідність методів програмування і використовуваних для них стандартів;**
  - **можливість побудови та тестування;**
  - **можливість експлуатації і супроводу.**

ISO/IEC 12207. 1.3. Розробка

1.3.8. Збірка програмних засобів. Розробник повинен:

- **Розробити план збірки для об'єднання програмних модулів і компонентів в програмний об'єкт. План повинен включати вимоги до випробувань (тестування), процедури тестування, контрольні дані, обов'язки виконавця і програму випробувань.**
- **Зібрати програмні модулі і компоненти і протестувати їх як продукти, розроблені відповідно до плану збірки.**
- **Забезпечити, щоб кожна збірка задовольняла вимогам до програмного об'єкту і щоб програмний об'єкт був повністю зібраний в результаті даної роботи.**
- **Документально оформити результати складання і тестування**

ISO/IEC 12207. 1.3. Розробка

1.3.8. Збірка програмних засобів.

Розробник повинен:

- **Розробити і документально оформити для кожної кваліфікаційної вимоги до програмного об'єкту - набір тестів, контрольних прикладів (вихідні і вихідні дані, критерії тестування), процедури випробувань для проведення кваліфікаційних випробувань програмних засобів.**
- **Забезпечити, щоб зібраний програмний об'єкт був готовий до кваліфікаційних випробувань.**

ISO/IEC 12207. 1.3. Розробка

1.3.9. Кваліфікаційні випробування програмних засобів. Розробник повинен:

- **Проводити кваліфікаційні випробування (тестування) на відповідність кваліфікаційним вимогам до програмного об'єкту.**
- **Забезпечити, щоб реалізація кожного встановленого вимоги до програмному об'єкту була перевірена на відповідність.**
- **Результати кваліфікаційних випробувань повинні бути документально оформлені.**
- **Повинен уточнити документацію користувача (при необхідності).**
- **Оцінити проект, запрограмований програмний об'єкт, проведені випробування, результати випробувань і документацію користувача.**

ISO/IEC 12207. 1.3. Розробка

1.3.10. Збірка системи. Розробник повинен:

- **Зібрати об'єкти програмної конфігурації в єдину систему разом з об'єктами технічної конфігурації, ручними операціями і, при необхідності, з іншими системами.**
- **Зібрана система повинна бути випробувана на відповідність встановленим вимогам. Результати повинні бути документально оформлені.**
- **Для кожного кваліфікаційного вимоги до системи розробити і документально оформити: склад випробувань і контрольних прикладів (вихідні та вихідні дані, критерії випробувань) та процедури проведення кваліфікаційних випробувань системи.**
- **Забезпечити, щоб зібрана система була готова до кваліфікаційних випробувань.**

ISO/IEC 12207. 1.3. Розробка

1.3.11. Кваліфікаційні випробування системи.

Розробник повинен:

- ❑Провести випробування у відповідності з кваліфікаційними вимогами, встановленими до системи.
- ❑Забезпечити, щоб реалізація кожного вимоги до системи була випробувана на відповідність встановленим значенням і щоб система була готова до постачання.
- ❑Документально оформити результати кваліфікаційних випробувань.
- ❑Оцінити систему за наступними критеріями
  - тестове покриття вимог до системи;
  - відповідність очікуваних результатів;
  - можливість експлуатації і супроводу.

ISO/IEC 12207. 1.3. Розробка

1.3.12. Забезпечення приймання програмних засобів. Розробник повинен:

- **Забезпечити проведення замовником оцінки готовності до приймання і приймальним випробуванням програмного продукту з крахуванням кваліфікаційних випробувань програмного продукту і кваліфікаційних випробувань системи.**
- **Результати оцінок готовності до приймання та приймальних випробувань повинні бути документально оформлені.**
- **Укомплектувати і поставити програмний продукт замовнику, дотримуючись умов договору.**
- **Забезпечити первинне і безперервне навчання і підтримку персоналу замовника.**

ISO/IEC 12207. 1.3. Розробка

1.3.2. Вимоги до системи

1.3.11. Випробування системи

1.3.3. Проектування системної архітектури

1.3.10. Збірка системи

„СІСТЕМНІ” РОБОТИ

1.3.4. Аналіз вимог до ПЗ

„ПРОГРАМНІ” РОБОТИ

1.3.9. Випробування програмних засобів

1.3.5. Проектування програмної архітектури

1.3.8. Збірка програмних засобів

1.3.6. Технічне проектування програмних засобів

1.3.7. ПРОГРАМУВАННЯ та ТЕСТУВАННЯ

ISO/IEC 12207. 1.3. Розробка

ISO/IEC 12207. 1.3.7. Життєвий цикл

Модель життєвого циклу (life cycle model):

структура, що складається з процесів, робіт і завдань, що включають в себе розробку, експлуатацію та супровід програмного продукту, що охоплює життя системи від встановлення вимог до неї до припинення її використання.

Виділяють наступні процеси:

- **Замовлення, поставка**
- **Аналіз**
- **Проектування**
- **Програмування**
- **Тестування**
- **Ввід в дію, експлуатація**

ISO/IEC 12207. 1.3.7. Життєвий цикл

Каскадна модель передбачає послідовне виконання всіх етапів проекту в строго фіксованому порядку.

Перехід на наступний етап означає повне завершення робіт на попередньому етапі.

ISO/IEC 12207. 1.3.7. Життєвий цикл

Каскадна модель

- фиксований набір стадій;
- кажна стадія закінчується документованим результатом;
- наступна стадія починається лише після закінчення попередньої.

Недоліки:

- негнучкість;
- фаза повинна бути завершена до переходу до наступної;
- набір фаз фіксований;
- ! НЕМОЖЛИВО (важко) реагувати на зміни вимог.

ISO/IEC 12207. 1.3.7. Життєвий цикл

Ітеративна (інкрементна) модель надає можливість повернень на попередні етапи.

Замовлення
Поставка

Аналіз вимог

Мета кожної ітерації – отримання працюючої версії програмної системи, що включає функціональність, визначену інтегрованим змістом усіх попередніх і поточної ітерації. На кожній ітерації продукт інкрементально нарощує функціональність. Результат фінальної ітерації → необхідний продукт.

ISO/IEC 12207. 1.3.7. Життєвий цикл

Ітеративна (інкрементна) модель

- Стадії повторюються неодноразово;

Недоліки:

- система часто погано структурована, проект «не прозорий»;
- після уточнення вимог відкидається частина раніше виконаної роботи;
- потрібні засоби для швидкого розроблення.

ISO/IEC 12207. 1.3.7. Життєвий цикл

Спіральна (еволюційна) модель: серія послідовних ітерацій.

ISO/IEC 12207. 1.3.7. Життєвий цикл

Спіральна модель:

- проект має контрольні точки – milestones;
- на кожному витку спіралі створюється прототип На виході – продукт, що відповідає вимогам користувачів.

Переваги:

- ранній аналіз можливостей повторного використання;
- наявні механізми досягнення параметрів якості;
- модель дозволяє контролювати джерела проектних робіт і відповідних витрат;
- модель дозволяє вирішувати інтегровані завдання системного розроблення, які охоплюють програмну і апаратну складові створюваного продукту.

Недоліки:

- після уточнення вимог відкидається частина раніше виконаної роботи;
- потрібні засоби для швидкого розроблення.

ПРОГРАМУВАННЯ НЕ Є ОСНОВНИМ ВИДОМ ДІЯЛЬНОСТІ ПРИ СТВОРЕННІ ПРОМИСЛОВОЇ СИСТЕМИ (ПРОГРАМНОГО ПРОДУКТУ)

ПРОГРАМУВАННЯ ≈ 15-20% ЧАСУ РОЗРОБКИ

Контрольні запитання

- Надайте опис етапів розвитку мов програмування. Поясніть основні переваги сучасного етапу – компонентного програмування.
- Визначте основні види сучасного програмного забезпечення. Надайте особливості сучасних систем ПЗ.
- Визначте поняття технології. Поясніть особливості технологій програмування. Надайте перелік компонентів технологічного процесу. Опишіть базові вимоги до технологічного процесу .

The END
Додаткова Лекція

Рекомендована ЛІТЕРАТУРА

- Програмування числових методів мовою Python [Електронний ресурс] : підручник /за ред. А. В. Анісімова. – Київ : ВПЦ «Київський університет», 2014. – 640 с. – Режим доступу : http://surl.li/ubzkt. – Назва з екрана.
- Програмування числових методів мовою Python [Електронний ресурс] : навч. посіб. / за ред. А. В. Анісімова. – Київ : ВПЦ «Київський університет», 2013. – 463 с. – Оежим доступу : http://surl.li/ucdzx. – Назва з екрана.
- Яковенко А.В. Основи програмування. Python. Частина 1 [Електронний ресурс]: підручник / А. В. Яковенко. – Київ : КПІім. Ігоря Сікорського, 2018. – 195 с. – Режим доступу : https://ela.kpi.ua/server/api/core/bitstreams/dbbe8ff5-11d7-4a92-918c-d1445c3d20a7/content. – Назва з екрана.

Рекомендована ЛІТЕРАТУРА

- Навчальний посібник з дисципліни «Технології розробки програмного забезпечення» для студентів спеціальності 123 «Комп'ютерна інженерія» [Електронний ресурс] / уклад. Л.М. Дегтярьова, П.М. Гроза, С.В. Сомов. – Полтава : ПолтНТУ, 2017. – 218 с. – Режим доступу : **http://reposit.pntu.edu.ua/handle/PoltNTU/4461**. – Назва з екрана.
- Lutz M. Learning Python [Electronic resource] / M. Lutz. - 5-th ed. – O'Reilly Media, Inc., 2013. - 1593 p. – Режим доступу : https://www.twirpx.com/file/1206702/. – Назва з екрана.
- Python Cookbook: Recipes for Mastering Python 3 [Electronic resource]. - 3rd edition. – O'Reilly and Associates, 2013. - 704 p. – Режим доступу : http://surl.li/ucdzq. – Назва з екрана.

Рекомендована ЛІТЕРАТУРА

- Інформаційні технології та моделювання бізнес-процесів / О.М. Томашевський, Г.Г. Цегелік, М.Б. Вітер, В.Ш. Дудук : навч. посіб. – Київ : ЦУЛ, 2012. – 296 с.
- Карпенко М.Ю. Технології створення програмних продуктів та інформаційних систем: навч. посіб. / М.Ю. Карпенко, Н.О. Манакова, І.О. Гавриленко. – Харків : ХНУМГ ім.О.М. Бекетова, 2017. – 93 с.
- Алексенко О.В. Технології програмування та створення програмних продуктів : конспект лекцій / О.В. Алексенко. – Суми : СумДУ, 2013. – 133 с.

Рекомендована ЛІТЕРАТУРА

- Jorgensen Paul C. Software testing. A Craftsman's Approach [Electronic resource] / Paul C. Jorgensen. - Fourth Edition. – NY: CRC Press, 2014. – 437 p. – Режим доступу : http://surl.li/ucdyz. – Назва з екрана.
- Vance S. Quality Code. Software Testing Principles, Practices and Patterns [Electronic resource] / S. Vance. – NY: Addison-Wesley, 2014. – 231 p. – Режим доступу : http://www.irbis-nbuv.gov.ua/cgi-bin/irbis_nbuv/cgiirbis_64.exe?C21COM=S&I21DBN=EC&P21DBN=&S21FMT=JwU_B&S21ALL=%28%3C.%3EU%3D%D0%97973-018.025$%3C.%3E%29&Z21ID=&S21SRW=AVHEAD&S21SRD=&S21STN=1&S21REF=10&S21CNR=20. – Назва з екрана.

Інформаційні ресурси

- ISTQB – International Software Testing Qualification Board [Електронний ресурс] : сайт. – Режим доступу : https://www.istqb.org/ – Назва з екрана.
- ISTQB – Glossary [Електронний ресурс] : сайт. – Режим доступу : https://glossary.istqb.org/en/search/. – Назва з екрана.
- GitHub Help [Електронний ресурс] : сайт. – Режим доступу : https://help.github.com/en. – Назва з екрана.
- Git Magic (Магія Git) [Електронний ресурс] // Ben Lynn. - Режим доступу : http://www-cs-students.stanford.edu/~blynn/gitmagic/intl/uk/ch10.html. – Назва з екрана.

Інформаційні ресурси

- Free Computer Science University [Електронний ресурс] : сайт. – Режим доступу : https://github.com/iteachmachines/Free-Computer-Science-University. – Назва з екрана.
- CS Books [Електронний ресурс]. – Режим доступу : https://github.com/AB1908/CS-Books. – Назва з екрана.
- A curated list about Python in Education [Електронний ресурс]. – Режим доступу : https://github.com/quobit/awesome-python-in-education#roadmaps. – Назва з екрана.
- Source Code for the Book Classic Computer Science Problems in Python [Електронний ресурс]. – Режим доступу : https://github.com/davecom/ClassicComputerScienceProblemsInPython. – Назва з екрана.