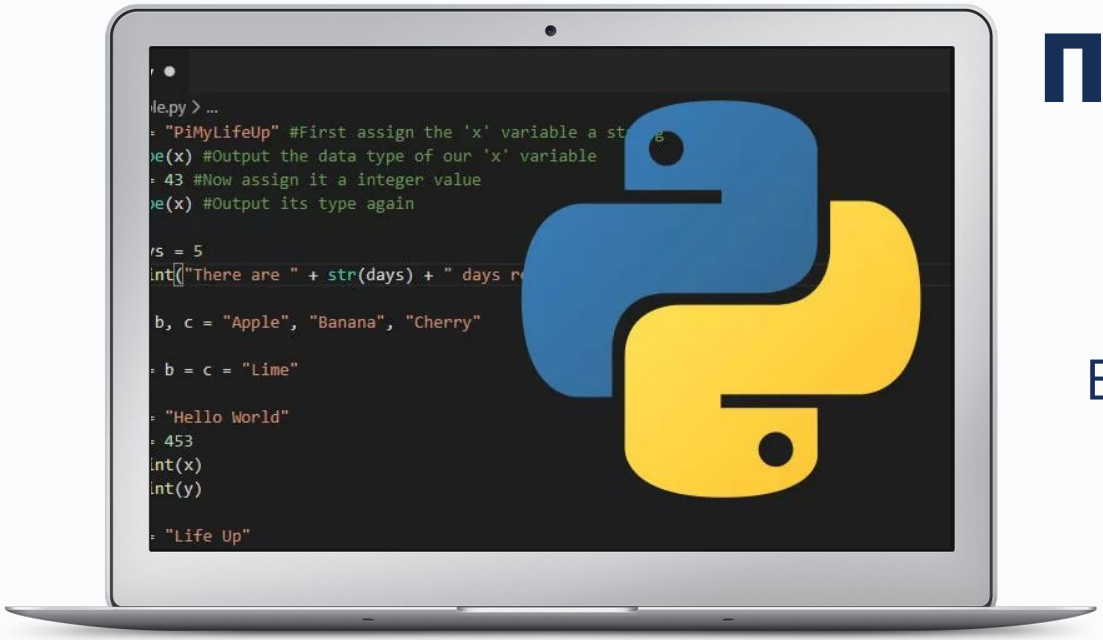
Become QA Auto

PROMETHEUS



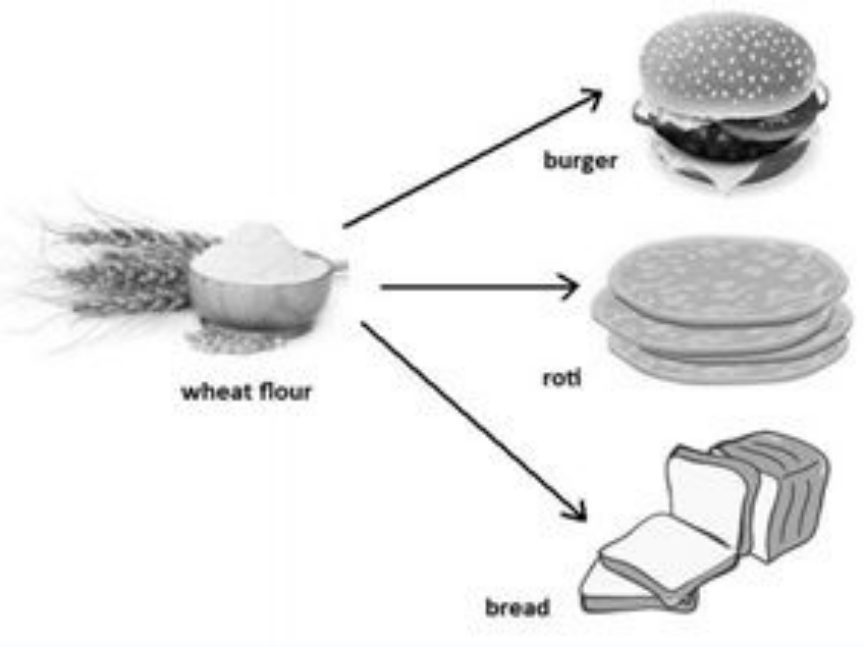
Поліморфізм

Бутенко Сергій

План лекції

- **Поняття поліморфізму**
- **Види поліморфізму**
- **Перевизначення методів як спосіб реалізації поліморфізму**

Поняття поліморфізму



Поліморфізм - це здатність будь-яких даних оброблятися в більш ніж одній формі.

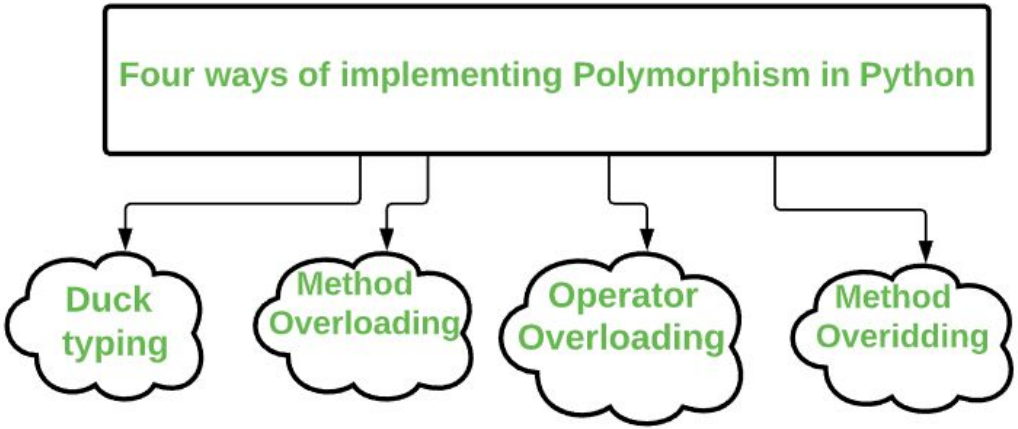
Саме слово складається з двох частин: **poly** - означає багато і **morphism** - означає форми.

Маючи пшеничне борошно, ми можемо готувати

- гамбургери,
- роті
- буханки хліба.

Це означає, що одне і те ж пшеничне борошно набуває різних їстівних форм.

Види поліморфізму в Python

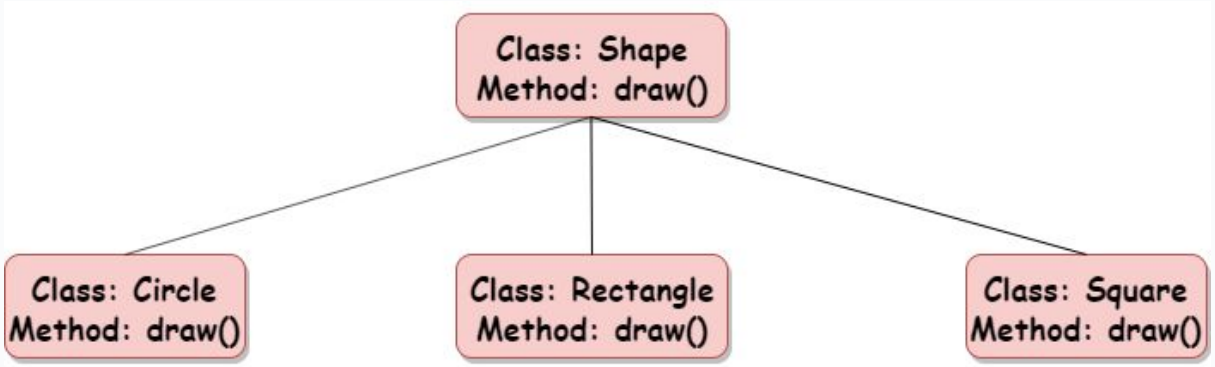


Існує **чотири способи** реалізації поліморфізму в Python:

- Філософія качиної типізації (Duck Typing)
- Перевантаження методу (Method Overloading)
- Перевантаження оператора (Operator Overloading)
- Перевизначення методу (Method Overriding)

Детальніше тут: https://www.gkindex.com/python-advanced/python-polymorphism.jsp

Перевизначення методів як спосіб реалізації поліморфізму



Створення в дочірньому класі методу з таким самим ім'ям, що і у відповідного методу батьківського класу, називається **перевизначенням методу**.

Суть перевизначення методу - в класі наслідникуу в перевизначеному методі робиться схожа дія, яка дещо відрізняється від того, що робиться в методі батьківського класу.

Приклад. Поліморфізм

shopPoly.py

PYTHON_BASICS > shopPoly.py > ...

```python
class ShopWorker:
    _count_workers=0
    def __init__(self, name1='', age1=0):
        self.name = name1
        self.__age = age1
        self.setting_id()
    def setting_id(self):
        ShopWorker._count_workers +=1
        self.id = ShopWorker._count_workers
    def working(self):
        print ('Виконую роботу')
    def __str__(self):
        str_out="Працівник "+str(self.id)+": "+self.name+" "+str(self.__age)
        str_out += " всіх працівників " + str(ShopWorker._count_workers)
        return str_out
    @staticmethod
    def info():
        print('В магазині працює: ', ShopWorker._count_workers, ' працівників')
    @classmethod
    def naming_shop(cls, name):
        cls.name_shop= name
        return cls.name_shop
    def get_age(self):
        return self.__age
    def set_age(self, new_age):
        if (new_age < 0):
            new_age= - new_age
        self.__age=new_age
```

Опис класу **ShopWorker**
Створення атрибуту класу **count_workers**
Конструктор класу

Виклик методу **setting_id** з конструктора
Опис методу **setting_id**
Ініціалізація полів **count_workers** та **id**

Опис методу **working**

Опис методу **__str__**

Статичний метод info()

Метод класу naming_shop(cls, name):

Гетер для атрибуту **__age**

Сетер для атрибуту **__age**
Перевірка параметра **new_age** на допустиме значення, та його коригування

Приклад. Поліморфізм

```python
class Seller (ShopWorker):
    def __init__(self, name1='', age1=0, cash1=0 ):
        super().__init__(name1, age1)
        self.cash=cash1
    def __str__(self):
        return super().__str__()+ " працює продавцем з готівкою "+ str(self.cash)
    def working(self):
        print ('Обслуговую покупців')
class  StoreManage(ShopWorker):
    def working(self):
        print ('Керую  магазином')
class ShopCleaner(ShopWorker):
    def working(self):
        print ('Прибираю магазин')

worker_one = ShopWorker('Іван', 25)
seller1= Seller('Оксана',28, 3456.50)
store_manager = StoreManage()
shop_cleaner = ShopCleaner()

worker_one.working()
seller1.working()
store_manager.working()
shop_cleaner.working()
```

```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE  JUPYTER        Python

Виконую роботу
Обслуговую покупців
Керую  магазином
Прибираю магазин
```

Опис класу **Seller**, що наслідує **ShopWorker**
Перевизначення конструктора батьківського класу

Перевизначення методу **__str__** в класі **Seller.**

Перевизначення методу **working** в класі **Seller.**

Опис класу **StoreManage**, що наслідує **ShopWorker**
Перевизначення методу **working** в класі **StoreManage.**

Опис класу **ShopCleaner**, що наслідує **ShopWorker**
Перевизначення методу **working** в класі **ShopCleaner.**

Створення об'єкту **worker_one**
Створення об'єкту **seller1**
Створення об'єкту **store_manager**
Створення об'єкту **shop_cleaner**

Виклик методу **working()** об'єктом **worker_one**
Виклик методу **working()** об'єктом **seller1**
Виклик методу **working()** об'єктом **store_manager**
Виклик методу **working()** об'єктом **shop_cleaner**

Результат роботи коду

Підсумки

Поліморфізм - це здатність будь-яких даних оброблятися в більш ніж одній формі.

Найбільш поширене використання поліморфізму в об’єктно-орієнтованому програмуванні відбувається при **перевизначенні методів** батьківського класу в дочірньому класі.

Існує чотири способи реалізації поліморфізму в Python:

- Філософія качиної типізації (Duck Typing)
- Перевантаження методу (Method Overloading)
- Перевантаження оператора (Operator Overloading)
- Перевизначення методу (Method Overriding)