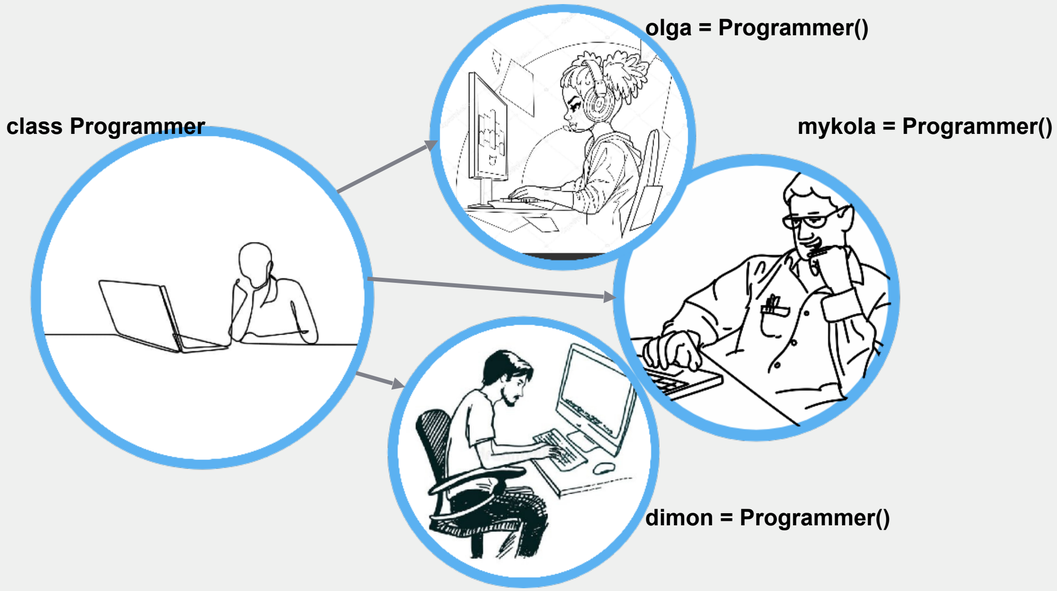
ООР: визначення і базове розуміння

- **об’єктно-орієнтоване програмування (ООП)** — це парадигма програмування , яка забезпечує засоби структурування програм таким чином, щоб властивості та поведінка об’єднувалися в окремі об’єкти .
Тобто - це просто ще одна методологія організації коду. Але завдяки своїй зручності вона втілена в багатьох сучасних мовах програмування і займає провідну позицію в сучасній розробці. Тому що це - зручно.

- **в центрі ООП - об’єкти,** які мають свої **атрибути**. Умовно атрибути поділяються на **властивості** (це іменовані поля з даними, кажуть що вони відображають стан об’єкта) і **методи** (функції які “прикріплені” до цього об’єкта - тобто находяться в його просторі імен, вони забезпечують дії - тобто поведінку).



ООП розрізняє дві основні дійові особи:

- **клас (class)** - це окремий об’єкт, який має свої власні властивості (іменовані поля даних) і “поведінку” (“вбудовані” в його простір імен функції). Основна особливість класу - він вміє створювати нові об’єкти (екземпляри).
- **екземпляр (instance)** - об’єкт який створений від якогось класу і може мати свої власні властивості (іменовані поля даних).

Тобто - в більшості випадків клас це те що описує загальні властивості і поведінку для кожного конкретного створеного за допомогою цього класу екземпляру. Ви вже багато разів з цим зтикались протягом нашого курсу: int - це клас, а числа 1, 2, 5, … - це екземпляри цього класу.

ООР: як створити клас і екземпляр.

- **синтаксис визначення класу:**

```
>>> class Programmer:
        pass
>>>
```

ми створили окремий клас Programmer. Він ще абсолютно “пустий” (за винятком стандартної функціональності, з якою ми будемо поступово знайомитись).

- **синтаксис визначення екземпляру відповідного класу:**

```
>>> olga = Programmer()
>>> type(olga)
<class '__main__.Programmer'>
>>> olga.__class__
<class '__main__.Programmer'>
>>> num = 10
>>> num.__class__
<class 'int'>
>>>
```

ми створили створили екземпляр olga класу Programmer. За допомогою функції type() можемо в цьому пересвідчитись. Зверніть увагу як відображається ім’я класу - в просторі імен модуля “__main__”

кожен об’єкт в пайтоні належить до якогось класу. І кожен об’єкт має спеціальний службовий атрибут “__class__” в якому зберігається посилання на батьківській клас. І будь-які знайомі нам об’єкти - також (наведено приклад для цілого числа). Таких “службових” атрибутів у об’єктів доволі багато, з частиною з них ми познайомимось. Доступ до них - за допомогою крапкової нотації.

OOP: методи класів.

- **поведінка: методи**

```
>>> class Programmer:
        def create_code(self):
            return “awesome code”

        def create_code_with_bugs(self):
            return “code with bugs”
>>>
```

якщо наші екземпляри класу програміст мають якусь спільну поведінку, то її можна визначити в класі Programmer. Наприклад - програмісти вміють писати код. Ми можемо створити прямо простірі імен класу функцію, яка буде реалізовувати цей функціонал (ігноруйте поки аргумент self - про це трохи пізніше). Таку функцію називають методом класу. Це дасть можливість викликати цю функцію за допомогою крапкової нотації від імені наших екземплярів (і від імені класу, звісно, також. Хоча це і роблять значно рідше.
Ну і програмісти не лише пишуть чудовий код, а і створюють баги). Тому додамо і такий метод.)

- **визначені в класі методи доступні в екземплярах:**

```
>>> olga = Programmer()
>>> olga.create_code()
'awesome code'
>>> olga.create_code_with_bugs()
'code with bugs'
>>>
```

Тепер об’єкти класу Programmer можуть реалізовувати поведінку, яка визначена в класі.
Точно таким чином всі цілі числа, наприклад, додаються одне до одного за правилами арифметика, а рядки - конкатенуються. Поведінку об’єкта визначено в класі, до якого він належить.

ООР: властивості класів.

- **дані: властивості класу**

```
>>> class Programmer:
        company = "Website creation company"
        def create_code(self):
            return "awesome code"

        def create_code_with_bugs(self):
            return "code with bugs"
>>>
```

Перед цім ми визначили загальну для всіх екземплярів поведінку - визначили в тілі класу методи (функції) і побачили що ми можемо їх викликати за допомогою крапкової нотації від імені екземплярів. Це стосується не тільки поведінки - тобто метдів. Це може відноситись і до даних. Такі поля називаються **властивостями класу.**
Якщо ми маємо якісь дані, які мають відношення до всіх екземплярів класу, який ми створюємо - тобто ми маємо загальну для всіх екземплярів інформацію, то ми можемо записати таку інформацію в тілі класу - і вона буде доступна за допомогою крапкової нотації як в об'єкті класу, так і в об'єктах екземплярів.

- **визначені в класі властивості доступні в екземплярах:**

```
>>> olga = Programmer()
>>> dimon = Programmer()
>>> olga.company
'Website creation company'
>>> dimon.company
'Website creation company'
>>> dimon.create_code_with_bugs()
'code with bugs'
```

Ця поля даних будуть доступні в усіх екземплярах класу за допомогою крапкової нотації.

ООР: властивості екземплярів.

- **дані: властивості екземпляру**

```
>>> olga.name = “Olga”
>>> olga.prg_lng = “python”
>>> dimon.name = “Dima”
>>> dimon.prg_lng = “js”
>>> olga.name
‘Olga’
>>> dimon.prg_lng
‘js’
```

наші екземпляри класу зараз ніяк не індивідуалізовані - вони мають все що вони отримали від класу. Але кожен екземпляр може мати свої властивості. Ми можемо додати їх “на льоту” - після створення екземпляру

```
>>> def init(inst, name, prg_lng):
        inst.name = name
        inst.prg_lng = prg_lng
>>>
>>> olga = Programmer()
>>> init(olga, “Olga”, “python”)
>>> olga.name
‘Olga’
```

Додавати так властивості до кожного екземпляру можливо, але - не зручно, особливо якщо властивотей багато і вони для кожного екземпляра є однаковими за суттю але різними за наповненням - кожен програміст має ім’я і мову програмування, але вони у всіх програмістів різні.
Ми можемо написати функцію, яка буде ініціалізувати екземпляр - приймати екземпляр і значення відповідних властивостей і записувати їх у відповідні поля.
Тобто - ми можемо створювати екземпляр і одразу його ініціалізувати його власними атрибутами. Такі атрибути називаються атрибути екземпляру і мають, як правило, власні значення для кожного окремого екземпляру. Як правило - це властивості - тобто екземпляри один від одного відрізняються якимось набором власних даних.
В нашому прикладі всі програмісти (екземпляри класу Programmer) можуть мати власне ім’я - і мову програмування: тобто власні значення для полів name і prg_lng.

ООР: конструктор екземпляру __init__.

- **конструктор екземпляру:**

```
>>> class Programmer:
        company = "Website creation company"

        def __init__(self, name, prg_lng):
            self.name = name
            self.prg_lng = prg_lng

        def create_code(self):
            return "awesome code"

        def create_code_with_bugs(self):
            return "code with bugs"
```

Майже завжди екземпляри класів мають свої властивості: коли ми описуємо користувача, то він має однаковий набір полів, але значення для кожного користувача свої: ім'я, email, дата народження і таке інше. Для класу Post будуть поля title з назвою і text з текстом посту. Але інформація в цих полях буде своєю для кожного посту. В нашому класі Programmer екземпляри можуть мати поля name і prg_lng з індивідуальною інформацією.
Тому існує "спеціальний" метод - **__init__** - який можна визначити в тілі класу, і він автоматично буде викликаний при створенні екземпляру класу - одразу після його створення. У якості першого аргумента йому автоматично передається сам цей екземпляр, а далі - бажані для ініціалізації екземпляра параметри. Фактично він працює саме так, як попередньо створена нами функція - лише викликається автоматично.

```
>>> olga = Programmer("Olga", "python")
>>> olga.name
'Olga'
>>> olga.prg_lng
'python'
>>>
```

Таким чином, завдяки визначенню **__init__** в тілі класу, ми можемо ініціювати наші екземпляри відповідними параметрами прямо під час їх створення.
Цей метод називається "конструктор екземпляра". Ще раз - спочатку створюється екземпляр, і, якщо існує визначений в класі метод __init__, він одразу автоматично викликається: першим аргументом йому передається сам цей екземпляр, а далі - описані при визначенні аргументи.

OOP: шлях пошуку атрибутів

```
>>> class Programmer:
        company = "Website creation company"

        def __init__(self, name, prg_lng):
            self.name = name
            self.prg_lng = prg_lng

        def create_code(self):
            return "awesome code"

        def create_code_with_bugs(self):
            return "code with bugs"
>>> olga = Programmer("Olga", "python")
>>> olga.name
'Olga'
>>> olga.prg_lng
'python'
>>> olga.position = "backend developer"
>>> olga.company
'Website creation company'
>>> olga.company = "New company"
>>> olga.company
'New company'
```

Треба розуміти, що клас і екземпляр класу - два різних об'єкта, кожен із своїм простором імен. В просторі імен класу знаходяться визначені в ньому імена, а в просторі імен екземпляра - визначені в ньому (або при ініціалізації, або пізніше).
Коли ми звертаємось за допомогою крапкової нотації від імені екземпляра до якогось імені - то спочатку інтерпретатор шукає його в просторі імен самого экземпляра і повертає якщо знайшов. Якщо не знайшов, то за допомогою службового атрибута __class__ звертається до простору імен класу і шукає там. Знайшов - повертає, ні - ми поговоримо про це пізніше, на цьому етапі вважаємо що генерує виключення.
Якщо ми визначимо в просторі імен екземпляра ім'я, яке співпадає з ім'ям в класі (в нашому прикладі - company) - ми не перевизначаємо нічого в класі, ми створюємо нове ім'я в просторі екземпляра. Таким чином ми втратимо прямий доступ до значення пов'язаного з цим ім'ям в класі, але лише для цього екземпляра. Всі інші - будуть мати доступ до імені класу, тому що в їх просторах не буде цього імені.