4. ОБЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ.

4.1. Основні поняття.

Багато сучасних мов підтримують кілька парадигм програмування (наприклад, директивне, функціональне, об'єктно-орієнтоване). Такі мови є змішаними. До них відносять і Python. Ключову роль в ООП відіграє множина об'єктів. Реальний світ складається з об'єктів та їхньої взаємодії між собою. В результаті взаємодії об'єкти можуть змінюватися самі або змінювати інші об'єкти. У світі умовно можна виділити різні системи, які реалізують певні цілі (змінюються з одного стану в інший). Наприклад, група на занятті – це система, яка складається з таких об'єктів, як діти, учитель, столи, комп'ютери, проектор тощо. У цієї системи можна виділити основну мету – збільшення частки знань дітей на якусь величину. Щоб домогтися цього, об'єкти системи повинні певним чином виконати взаємодію між собою.

Необхідно розуміти різницю між програмою написаною на основі структурного «стилю» і програмою в «стилі» ООП. У першому випадку, на перший план виходить логіка, розуміння послідовності виконання виразів (дій) для досягнення цілей. У другому – важливо системне мислення, вміння бачити систему в цілому, з одного боку, і розуміння ролі її частин (об'єктів), з іншого.

Далі наведено основні принципи, які описують ООП:

1) об'єктно-орієнтована програма складається з об'єктів, які посилають один одному повідомлення;

2) кожний об'єкт може складатися з інших об'єктів (а може і не складатися);

3) кожний об'єкт належить певному класу (типу), який задає поведінку об'єктів, створених на його основі.

Клас – це опис об'єктів певного типу. На основі класів створюють об'єкти. Може існувати множина об'єктів, які належать одному класу. З іншого боку, може існувати клас без об'єктів, реалізованих на його основі. Програма, написана з використанням парадигми ООП, повинна складатися з: об'єктів, класів (опису об'єктів), взаємодій об'єктів між собою, в результаті яких змінюються їх властивості. Об'єкт в програмі можна створити лише на основі будь-якого класу. Тому, ООП має розпочинатися з проектування і створення класів. Класи можуть бути розташовані або спочатку коду програми, або імпортуватися з інших файлів – модулів (також на початку коду).

Створення класів. Для створення класів передбачена інструкція *class*, яка складається з рядка заголовка і тіла. Заголовок складається з ключового слова *class*, імені класу і, можливо, назв суперкласів в дужках. Суперкласи можуть бути відсутніми, в такому випадку дужки не потрібні. Тіло класу складається з блоку різних інструкцій. Тіло повинно мати відступ (як і будь-які вкладені конструкції в Python). Схематично клас можна подати таким чином:

class ІМ‘Я_КЛАСУ:
змінна = значення ….
def ІМ‘Я_МЕТОДА(self, ...):
self.змінна = значення …

У заголовку після імені класу можна вказати суперкласи (в дужках), а методи можуть бути більш складними. Методи в класах – це звичайні функції,за одним винятком: вони мають один обов'язковий параметр – *self*, який потрібен для зв'язку з конкретним об'єктом. Атрибути класу – це імена змінних поза функцій і імена функцій. Вони успадковуються всіма об'єктами, створеними на основі даного класу, і забезпечують властивості і поведінку об'єкта. Об'єкти можуть мати атрибути, які створюються в тілі методу, якщо даний метод буде викликано для конкретного об'єкта.

Створення об'єктів. Об'єкти створюються так:

змінна = ІМ‘Я_КЛАСУ()

Після такої інструкції у програмі з'являється об'єкт, доступ до якого можна отримати за ім'ям змінної, пов'язаної з ним. При створенні об'єкт отримує атрибути його класу (він має характеристики, визначені у його класі). Кількість об'єктів, які можна створити на основі певного класу, не обмежена. Об'єкти одного класу мають схожий набір атрибутів, а значення атрибутів можуть бути різними (вони схожі, але індивідуально різняться).

Self: Методи класу - це невеликі програми, призначені для роботи з об'єктами. Методи можуть створювати нові властивості (дані) об'єкта, змінювати існуючі, виконувати інші дії над об'єктами. Методу необхідно «знати», дані якого об'єкта йому потрібно буде обробляти. Для цього йому в якості першого (а іноді і єдиного) аргументу передається ім'я змінної, пов'язаної з об'єктом. Щоб в описі класу вказати об'єкт, який передається в подальшому, використовують параметр *self*. Виклик методу для конкретного об'єкта в основному блоці програми виглядає таким чином: *ОБ'ЄКТ.ІМ‘Я_МЕТОДА(...)*, де *ОБ'ЄКТ* – змінна, пов‘язана з ним. Цей вираз перетворюється в класі, до якого відноситься об‘єкт, в *ІМ‘Я_МЕТОДА(ОБ'ЄКТ, ...)* – замість параметра *self* підставляють конкретний об'єкт. Атрибути екземпляра створюються шляхом присвоювання значень атрибутам об'єкта екземпляра. Зазвичай вони створюються всередині методів класу, в інструкції *class* – присвоєнням значень атрибутам аргументу *self* (який завжди є імовірним екземпляром). Можна створювати атрибути за допомогою операції присвоєння в будь-якому місці програми, де доступне посилання на екземпляр, навіть за межами інструкції *class*.

Конструктор. Зазвичай всі атрибути екземплярів ініціалізують в конструкторі *__init__*. Метод конструктора *__init__* використовують для

встановлення початкових значень атрибутів екземплярів і виконання інших початкових операцій (це - звичайна функція, яка підтримує і можливість визначення значень аргументів за замовчуванням, і передачу іменованих аргументів).

4.2.Абстракція і декомпозиція.

Абстракція в ООП дозволяє скласти з даних і алгоритмів обробки цих даних об'єкти, не беручи до уваги несуттєві (на деякому рівні) з точки зору складеної інформаційної моделі деталей. Таким чином, програма піддається *декомпозиції* на частини "дозованої" складності. Окремий об'єкт, навіть разом з сукупністю його зв'язків з іншими об'єктами, людиною сприймається легше (саме так він звик оперувати в реальному світі), ніж щось неструктуроване і монотонне.

Перед тим як почати написання, навіть самої простої ,об'єктної програми, необхідно провести аналіз предметної області, для того щоб виявити в ній класи об'єктів.

При виділенні об'єктів необхідно абстрагуватися від більшості притаманних їм властивостей і сконцентруватися на властивостях, які є значущими для завдання.

Вдала декомпозиція є трудозатратною. Від неї залежать не тільки кількісні характеристики коду (швидкодія, яку займає пам'ять), але і трудомісткість подальшого розвитку і супроводу.

4.3. Об'єкти.

В даному методичному посібнику *об'єкти* Python зустрічалися багато раз: адже кожне число,рядок, функція, модуль і т.п. - це *об'єкти*. Деякі вбудовані об'єкти мають в Python синтаксичну підтримку (для задання литералов):

```
a = 3
b = 4.0
c = a + b
```

Спочатку ім'я "a" зв'язується в локальному просторі імен з об'єктом-числом 3 (ціле число). Потім "b" зв'язується з об'єктом-числом 4.0 (число з плаваючою точкою). Після цього над об'єктами 3 і 4.0 виконується операція додавання, і ім'я "c" зв'язується з кінцевим об'єктом.

До речі, операціями, в основному, будуть називатися методи, які мають в Python синтаксичну підтримку. Те ж саме можна записати як:

```
c = a.__add__(b)
```

Тут *__add __ ()* - метод об'єкта *a*, який реалізує операцію + між цим та іншим об'єктом.

Дізнатися набір методів для деякого об'єкта можна за допомогою вбудованої функції *dir()*:

>>> dir(a)
['__abs__', '__add__', '__and__', '__class__', '__cmp__', '__coerce__',
'__delattr__', '__div__', '__divmod__', '__doc__', '__float__',
'__floordiv__', '__getattribute__', '__getnewargs__', '__hash__',
'__hex__', '__init__', '__int__', '__invert__', '__long__',
'__lshift__', '__mod__', '__mul__', '__neg__', '__new__',
'__nonzero__', '__oct__', '__or__', '__pos__', '__pow__',
'__radd__', '__rand__', '__rdiv__', '__rdivmod__', '__reduce__',
'__reduce_ex__', '__repr__', '__rfloordiv__', '__rlshift__',
'__rmod__', '__rmul__', '__ror__', '__rpow__', '__rrshift__',
'__rshift__', '__rsub__', '__rtruediv__', '__rxor__',
'__setattr__', '__str__', '__sub__', '__truediv__', '__xor__']

Тут варто вказати на ще одну особливість Python. Не тільки інфіксні операції, але і вбудовані функції очікують наявності деяких методів у об'єкта:

abs(c)

Функція abs () насправді використовує метод переданого їй об'єкта:

c.__abs__()

Об'єкти з'являються в результаті виконання функцій-фабрик або конструкторів класів, а закінчують своє існування при видаленні останнього посилання на об'єкт. Оператор del видаляє ім'я (а значить, і одне посилання на об'єкт) з простору імен:

a = 1
 # ...
del a
 # ім'я більше не існує

4.4. Типи і класи.

Тип визначає область допустимих значень об'єкта і набір операцій

надним. В ООП тип тісно пов'язаний з поведінкою - діями об'єкта, що складаються в зміні внутрішнього стану і викликами методів інших об'єктів.

Раніше в мові Python вбудовані типи даних не були екземплярами класу, тому вважалося, що це були просто об'єкти певного типу. Тепер ситуація змінилася, і об'єкти вбудованих типів мають класи, до яких вони належать. Таким чином, тип і клас в Python стають синонімами.

Інтерпретатор мови Python завжди може сказати, до якого типу належить об'єкт. Однак з точки зору застосування об'єкта в операції його приналежність до класу не грає вирішальної ролі: набагато важливіше, які методи підтримує об'єкт.

Екземпляри класів можуть з'являтися в програмі не тільки з литералов або в результаті операцій. Зазвичай для отримання об'єкта класу досить викликати конструктор цього класу з деякими параметрами. Об'єкт-клас, як і об'єкт функції, може бути викликаний. Це і буде викликом конструктора:

```
>>> import sets
>>> s = sets.Set([1, 2, 3])
```

У цьому прикладі модуль sets містить визначення класу Set. Викликається конструктор цього класу з параметром [1, 2, 3]. В результаті з ім'ям s буде зв'язаний об'єкт-множина з трьох елементів 1, 2, 3.

Слід зауважити, що, крім конструктора, певні класи мають і деструктор - метод, який викликається при знищенні об'єкта. В мові Python об'єкт знищується в разі видалення останнього посилання на нього або в результаті сбору сміття, якщо об'єкт виявився в невживаному циклі посилань. Так як Python сам управляє розподілом пам'яті, деструктори в ньому потрібні дуже рідко. Зазвичай в тому випадку, коли об'єкт управляє ресурсом, який потрібно коректно повернути в певний стан.

Ще один спосіб отримати об'єкт деякого типу - використання функцій-фабрик. За синтаксису виклик функції-фабрики не відрізняється від виклику конструктора класу.

Визначення класу

Нехай в ході аналізу даної предметної області необхідно визначити клас Граф. Граф - це безліч вершин і набір ребер, який попарно з'єднує ці вершини. Над графом можна проробляти операції, такі як додавання вершини, ребра, перевірка наявності ребра в графі і т.п. Мовою Python визначення класу може виглядати так:

```
class G:
    def __init__(self, V, E):
```

```
        self.vertices = set (V)
        self.edges = set (E)
    def add_vertex(self, v):
        self.vertices.add(v)
    def add_edge(self, v3):
        v1, v2 = v3
        self.vertices.add(v1)
        self.vertices.add(v2)
        self.edges.add((v1, v2))
    def has_edge(self, v3):
        v1, v2 = v3
        return ((v1, v2) in self.edges)
    def __str__(self):
        return ("%s; %s" % (self.vertices, self.edges))
```

Використовувати клас можна наступним чином:

```
g = G([1, 2, 3, 4], [(1, 2), (2, 3), (2, 4)])

print (g)
g.add_vertex(5)
g.add_edge((5,6))
print (g.has_edge((1,6)))
print (g)
```

```
{1, 2, 3, 4}; {(1, 2), (2, 3), (2, 4)}
False
{1, 2, 3, 4, 5, 6}; {(1, 2), (5, 6), (2, 3), (2, 4)}
```

Конструктор класу має спеціальне ім'я *__init__*. (Деструктор тут не потрібен, але він би мав ім'я *__del__*.) Методи класу визначаються в просторі імен класу. В якості першого формального аргументу методу прийнято використовувати *self*. Крім методів в об'єкті класу є два атрибути: *vertices* (вершини) і *edges* (ребра). Для представлення об'єкту *G* у вигляді строки використовується спеціальний метод *__str __* ().

Належність класу можна з'ясувати за допомогою вбудованої функції *isinstance ():*

```
print (isinstance (g, G))
```

4.5. Успадкування, інкапсуляція, поліморфізм.

Ідеї (принципи) ООП. Виділяють такі основні ідеї ООП як успадкування,

інкапсуляція і поліморфізм:

1) *Успадкування*. Можливість виділяти загальні властивості і методи класів в один клас верхнього рівня (батьківський). Класи, які мають загального батька, різняться між собою за рахунок включення до їх складу різних додаткових властивостей (атрибутів) і методів.

2) *Інкапсуляція*. Атрибути (властивості) і методи класу класифікують на доступні зовні (опубліковані) і недоступні (захищені). Захищені атрибути можна змінити, перебуваючи поза класом. Опубліковані атрибути також називають інтерфейсом об'єкта, тому що за їх допомогою з об'єктом можна взаємодіяти. Інкапсуляція покликана забезпечити надійність програми, тому що змінити істотні для існування об'єкта атрибути стає неможливо.

3) *Поліморфізм*. Поліморфізм має на увазі заміщення атрибутів, описаних раніше в інших класах: ім'я атрибута залишається колишнім, а реалізація вже іншою. Поліморфізм дозволяє адаптувати класи, залишаючи при цьому одинінтерфейс взаємодії.

Переваги ООП. Серед них можна виділити:

1. Використання одного і того ж програмного коду з різними даними. Класи дозволяють створювати множину об'єктів, кожен з яких має власні значення атрибутів. Немає потреби вводити множину змінних (об'єкти отримують в своє розпорядження індивідуальний простір імен). Простір імен конкретного об'єкта формується на основі класу, від якого він був створений, а також на основі усіх батьківських класів даного класу.

2. Успадкування і поліморфізм дозволяють не писати новий код, а налаштовувати вже існуючий, за рахунок додавання і перевизначення атрибутів. Це веде до скорочення обсягу вихідного коду. ООП дозволяє скоротити час на написання вихідного коду, проте ООП завжди передбачає велику роль попереднього аналізу предметної області, попереднього проектування. Від правильності розв‘язків на цьому попередньому етапі залежить куди більше, ніж від безпосереднього написання вихідного коду.

Особливості ООП в Python. У порівнянні з іншими поширеними мовами програмування у Python можна виділити такі особливості, пов'язані з ООП:

1) *Будь-які дані (значення) – це об'єкти*. Число, рядок, список, масив і ін.– все є об'єктом. Бувають об'єкти вбудованих класів (ті, що перераховані в попередньому реченні), а бувають об'єкти класів користувача (їх створює програміст). Для єдиного механізму взаємодії передбачені методи перезавантаження операторів .

2) *Клас – це об'єкт з власним простором імен*. Тому правильніше було вживати замість слова «об'єкт», слово «екземпляр». І говорити «екземпляр об'єкта», маючи під цим на увазі створений на основі класу саме об'єкт, і «екземпляр класу», маючи на увазі сам клас як об'єкт.

3) *Інкапсуляції в Python не приділяється особливої уваги*. В інших мовах

програмування зазвичай не можна отримати безпосередньо доступ до властивості, описаного в класі. Для його зміни може бути передбачений спеціальний метод. В Python це легко зробити, звернувшись до властивості класу із зовні. Незважаючи на це в Python передбачені спеціальні способи обмеження доступу до змінних в класі.