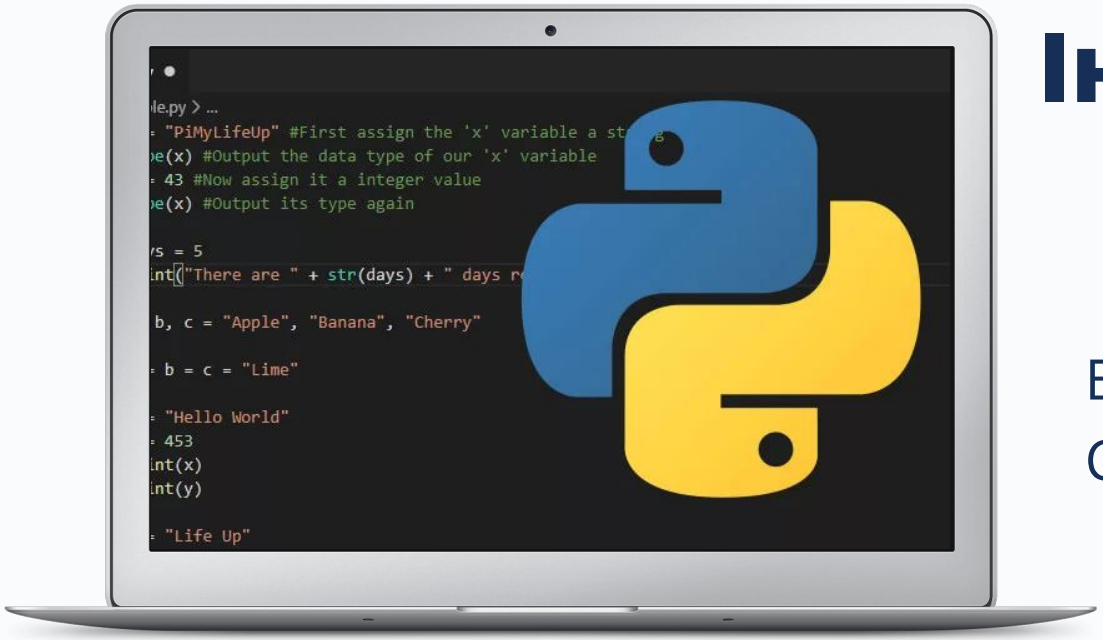
# Become QA Auto

PROMETHEUS




# Інкапсуляція

Бутенко
Сергій

# План лекції

- Поняття інкапсуляції
- Сетери та гетери
- Приховування атрибутів
- Приховування методів

# Поняття інкапсуляції


**Інкапсуляція -** це властивість системи, що дозволяє об'єднати дані і методи, що працюють з ними, в класі і приховати деталі реалізації від користувача.

**Інтерфейс класу (API)** – це набір загальнодоступних полів та методів.

# Приховування атрибутів

**Права доступу до окремих елементів:**

- властивості і методи можуть бути **закритими**, тобто доступними тільки всередині класу.
- дані та методи можуть бути **відкриті** - тоді до цих методів та даних можуть звертатися методи інших класів.

В Python механізму модифікаторів доступу не існує. Для приховування атрибутів в Python використовують домовленості:

- **символ підкреслення** перед атрибутом - то цей атрибут призначено виключно для внутрішнього використання (ніяк не захищено).
- **два символи підкреслення** перед атрибутом - це позначення приватного поля (механізм «**`name mangling`**»)

# Приклад 1. Приховування атрибутів

shopEncaps.py

PYTHON_BASICS > shopEncaps.py > ShopWorker > __str__

```
1   class ShopWorker:
2       count_workers=0
3       def __init__(self, name1='', age1=0):
4           self.name = name1
5           self.age = age1
6           self.setting_id()
7       def setting_id(self):
8           ShopWorker.count_workers +=1
9           self.id = ShopWorker.count_workers
10      def add_year(self):
11          self.age+=1
12      def __str__(self):
13          str_out="Працівник "+str(self.id)+": "+self.name+" "+str(self.age)
14          str_out += " всіх працівників " + str(ShopWorker.count_workers)
15          return str_out
16      @staticmethod
17      def info():
18          print('В магазині працює: ', ShopWorker.count_workers, ' працівників')
19      @classmethod
20      def naming_shop(cls, name):
21          cls.name_shop= name
22          return cls.name_shop
23
24  ShopWorker.info()
25
26  worker_one = ShopWorker('Іван', 25)
27  print (worker_one)
28  worker_one.add_year()
29  print (worker_one)
30  ShopWorker.naming_shop('Fara')
31  print ('Назва магазину: ',ShopWorker.name_shop)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER Python

```
В магазині працює:  0  працівників
Працівник 1: Іван 25 всіх працівників 1
Працівник 1: Іван 26 всіх працівників 1
Назва магазину:  Fara
```

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 **add_year** для збільшення віку об'єкта

Опис методу **__str__**

Статичний метод info()

Метод класу naming_shop(cls, name):

Вивід атрибуту класу **ShopWorker**

Створення об'єкту **worker_one**, вивід атрибутів об'єкту **worker_one,** виклик методу збільшення віку для об'єкта **worker_one,** повторний вивід атрибутів об'єкту **worker_one**

Виклик методу класу **naming_shop** через назву класу
Вивід атрибута класу **name_shop**

Результат роботи коду

# Приклад 1. Приховування атрибутів

shopEncaps.py

PYTHON_BASICS > shopEncaps.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 add_year(self):
        self.__age+=1
    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

ShopWorker.info()

worker_one = ShopWorker('Іван', 25)
print(worker_one)
worker_one.add_year()
print(worker_one)
ShopWorker.naming_shop('Fara')
print ('Назва магазину: ',ShopWorker.name_shop)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER Python

```
В магазині працює:  0  працівників
Працівник 1: Іван 25 всіх працівників 1
Працівник 1: Іван 26 всіх працівників 1
Назва магазину:  Fara
```

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 **add_year** для збільшення віку об'єкта

Опис методу **__str__**

Статичний метод info()

Метод класу naming_shop(cls, name):

Вивід атрибуту класу **ShopWorker**

Створення об'єкту **worker_one**, вивід атрибутів об'єкту **worker_one,** виклик методу збільшення віку для об'єкта **worker_one,** повторний вивід атрибутів об'єкту **worker_one**

Виклик методу класу **naming_shop** через назву класу
Вивід атрибута класу **name_shop**

Результат роботи коду

# Приклад 1. Приховування атрибутів

shopEncaps.py

PYTHON_BASICS > shopEncaps.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 add_year(self):
        self.__age+=1
    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

print(ShopWorker._count_workers)

worker_one = ShopWorker('Іван', 25)
print(worker_one.__age)
worker_one.add_year()
print(worker_one)
ShopWorker.naming_shop('Fara')
print ('Назва магазину: ',ShopWorker.name_shop)
```

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE JUPYTER Python

```
0
Traceback (most recent call last):
  File "d:/PYTHON_BASICS/shopEncaps.py", line 27, in <module>
    print(worker_one.__age)
AttributeError: 'ShopWorker' object has no attribute '__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 **add_year** для збільшення віку об'єкта

Опис методу **__str__**

Статичний метод info()

Метод класу naming_shop(cls, name):

Вивід атрибуту класу **ShopWorker**

Створення об'єкту **worker_one**, вивід атрибутів об'єкту **worker_one**, виклик методу збільшення віку для об'єкта **worker_one**, повторний вивід атрибутів об'єкту **worker_one**

Виклик методу класу **naming_shop** через назву класу.
Вивід атрибута класу **name_shop**

Результат роботи коду

# Сетери та гетери

**Спеціальні відкриті методи:**

- **Гетер** (від англійського get - «отримувати») - метод, який повертає значення поля.
- **Сетер** (від англійського set - «встановлювати») - метод, який встановлює значення поля.

Синтаксис опису гетера:

```
def get_<ім'я атрибута>(self):
    return self.<ім'я атрибута>
```

Виклик за межами класу

```
<ім'я об'єкта>.get_<ім'я атрибута>()
```

Синтаксис опису сетера:

```
def set_<ім'я атрибута>(self,<параметр>):
     self.<ім'я атрибута>=<параметр>
```

Виклик за межами класу:

```
<ім'я об'єкта>.set_<ім'я атрибута>(<параметр>)
```

# Приклад 2. Сетер та гетер для поля __age

shopEncaps1.py

PYTHON_BASICS > shopEncaps1.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 add_year(self):
        self.__age+=1
    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
```

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 **add_year** для збільшення віку об'єкта

Опис методу **__str__**

Статичний метод info()

Метод класу naming_shop(cls, name):

# Приклад 2. Сетер та гетер для поля __age

shopEncaps1.py

PYTHON_BASICS > shopEncaps1.py > ShopWorker > set_age

```
23      def get_age(self):
24          return self.__age
25      def set_age(self, new_age):
26          self.__age=new_age
27
28  ShopWorker.info()
29
30  worker_one = ShopWorker('Іван', 25)
31
32  print(worker_one.get_age())
33  worker_one.set_age(-44)
34  print(worker_one.get_age())
35  ShopWorker.naming_shop('Fara')
36  print ('Назва магазину: ',ShopWorker.name_shop)
```

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE JUPYTER Python

```
В магазині працює:  0  працівників
25
-44
Назва магазину:  Fara
```

Гетер для атрибуту **__age**

Сетер для атрибуту **__age**

Вивід атрибуту класу **ShopWorker**

Створення об'єкту **worker_one**,

Вивід атрибуту **age** об’єкту через виклик гетера
Виклик сетера для атрибуту **__age**
Повторний вивід атрибуту **__age** через гетер
Виклик методу класу **naming_shop** через назву класу
Вивід атрибута класу **name_shop**

Результат роботи коду

# Приклад 3. Сетер з умовою для поля __age

shopEncaps2.py

PYTHON_BASICS > shopEncaps2.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 add_year(self):
        self.__age+=1
    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
```

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 **add_year** для збільшення віку об'єкта

Опис методу **__str__**

Статичний метод info()

Метод класу naming_shop(cls, name):

# Приклад 3. Сетер з умовою для поля __age

shopEncaps2.py

PYTHON_BASICS > shopEncaps2.py > ...

```
23      def get_age(self):
24          return self.__age
25      def set_age(self, new_age):
26          if (new_age < 0):
27              new_age= - new_age
28          self.__age=new_age
29
30  ShopWorker.info()
31
32  worker_one = ShopWorker('Іван', 25)
33
34  print(worker_one.get_age())
35  worker_one.set_age(-44)
36  print(worker_one.get_age())
37  ShopWorker.naming_shop('Fara')
38  print ('Назва магазину: ',ShopWorker.name_shop)
```

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE JUPYTER Python

```
В магазині працює:  0  працівників
25
44
Назва магазину:  Fara
```

Гетер для атрибуту **__age**

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

Вивід атрибуту класу **ShopWorker**

Створення об'єкту **worker_one**,

Вивід атрибуту **age** об’єкту через виклик гетера
Виклик сетера для атрибуту **__age**
Повторний вивід атрибуту **__age** через гетер
Виклик методу класу **naming_shop** через назву класу
Вивід атрибута класу **name_shop**

Результат роботи коду

# Приховування методів

**Для приховування методів в Python також використовують домовленості:**

- символ підкреслення перед методом - то цей метод призначено виключно для внутрішнього використання (ніяк не захищено).
- два символи підкреслення перед атрибутом - це позначення приватного методу (механізм «name mangling»)

**Не слід оголошувати свої власні (нестандартні) методи з іменами, які починаються і закінчуються двома знаками підкреслення.**

# Підсумки

- **Інкапсуляція** - це властивість системи, що дозволяє об'єднати дані і методи, що працюють з ними, в класі і приховати деталі реалізації від користувача.
- В Python механізму модифікаторів доступу не існує.
- **Символ підкреслення** перед іменем атрибуту чи методу - означає що цей елемент призначено виключно для внутрішнього використання (ніяк не захищено).
- **Два символи підкреслення** перед іменем атрибуту чи методу - це позначення приватного елементу (механізм «name mangling»)
- **Гетер** (від англійського get - «отримувати») - метод, який повертає значення прихованого поля.
- **Сетер** (від англійського set - «встановлювати») - метод, який встановлює значення прихованого поля.