PODSTAWY PROGRAMOWANIA W JĘZYKU PYTHON

OBIEKTOWOŚĆ 2

ALEKSANDRA BARTNIK | 06.05.2024

POLE STATYCZNE

Pole statyczne -pole tworzone w klasie poza konstruktorem i jest wspólne dla wszystkich obiektów klasy.

```
class Licznik:
    ile = 0                     # pole statyczne
    def __init__(self):         # konstruktor
        Licznik.ile += 1        # odwołanie do pola statycznego
        self.ktory = Licznik.ile
        print(f"To jest obiekt nr {Licznik.ile}")
```

DOSTĘP DO PÓL

Pole publiczne - dostęp mają wszyscy;

Pole chronione - dostęp mają klasy dziedziczące;

Pole prywatne - dostęp ma tylko ta klasa.

```
class Test:
    def __init__(self):
        self.publiczne, self._chronione, self.__prywatne = 1, 2, 3
```

DOSTĘP DO PÓL

```
class Test:
    def __init__(self):
        self.publiczne, self._chronione, self.__prywatne = 1, 2, 3


test = Test()
print(test.publiczne)
print(test._chronione)
print(test.__prywatne) # to zwróci nam błąd
print(test._Test__prywatne) # ale wciąż możemy się odwołać przez klasę
```

Możemy też zmodyfikować pole prywatne ustwiając setter w klasie lub wypisać pole prywatne ustawiając getter w klasie.

DEKORATORY

Dekoratory- funkcje, które pozwalają na modyfikację lub rozszerzanie funkcjonalności innych funkcji lub klas. Działają one poprzez przyjęcie funkcji jako argumentu i zwrócenie innej funkcji.

```
#Output
before Execution
Inside the function
after Execution
Sum = 3
```

```
def hello_decorator(func):
    def inner1(*args, **kwargs):
        print("before Execution")
        returned_value = func(*args, **kwargs)
        print("after Execution")
        return returned_value

    return inner1



@hello_decorator
def sum_two_numbers(a, b):
    print("Inside the function")
    return a + b
print("Sum =", sum_two_numbers(1, 2))
```

DEKORATORY

```
def decor1(func):
    def inner():
        x = func()
        return x * x
    return inner

def decor(func):
    def inner():
        x = func()
        return 2 * x
    return inner
```

```
@decor1
@decor
def num():
    return 10

@decor
@decor1
def num2():
    return 10

print(num())
print(num2())
```

<=> decor1(decor(num))

#Output
400

<=>decor(decor1(num2))

#Output
200