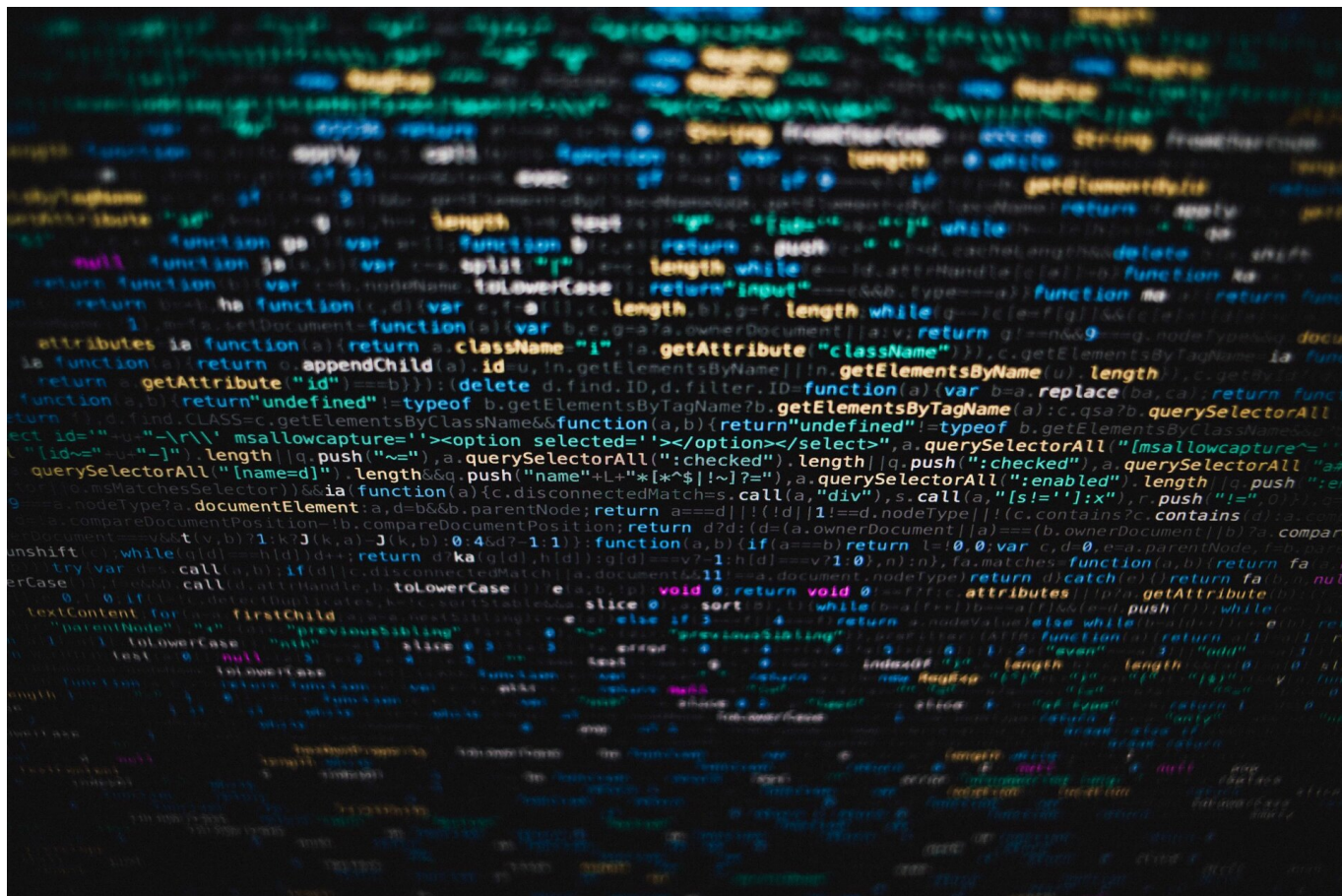
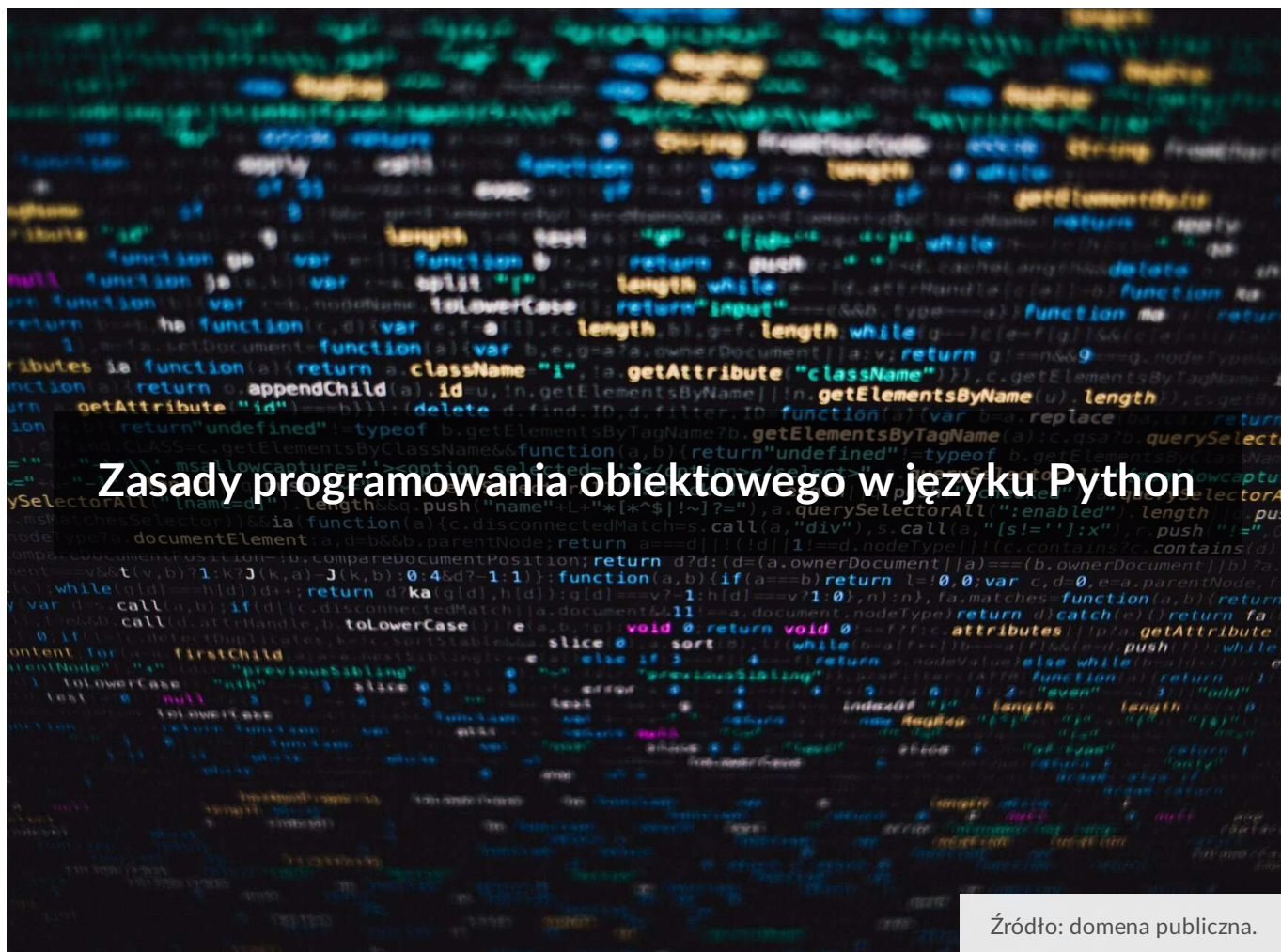




Zasady programowania obiektowego w języku Python

- Wprowadzenie
- Przeczytaj
- Prezentacja multimedialna
- Sprawdź się
- Dla nauczyciela



Zasady programowania obiektowego w języku Python

Źródło: domena publiczna.

Wiemy już, że programowanie obiektowe opiera się na kilku zasadach. Czym są abstrakcja, dziedziczenie, polimorfizm i hermetyzacja dowiedzieliśmy się w e-materiale [Zasady programowania obiektowego](#).

W tym e-materiale zajmiemy się realizacją zasad programowania obiektowego w języku Python.

Ciekawi cię, jak wyglądają implementacje w innych językach programowania? Możesz się z nimi zapoznać w dwóch pozostałych e-materiałach z tej serii:

- [Zasady programowania obiektowego w języku C++](#),
- [Zasady programowania obiektowego w języku Java](#).

Twoje cele

- Prześledzisz zaawansowane zasady programowania obiektowego w języku Python.
- Zaimplementujesz definicje klas wykorzystujących techniki programowania obiektowego.
- Stworzysz program wykorzystujący techniki programowania obiektowego i definicje klas.

Przeczytaj

Hermetyzacja

Hermetyzacja oznacza ukrywanie implementacji. W języku Python polega ona na oznaczeniu wybranych metod bądź pól, które nie powinny być używane poza klasą, w której są zdefiniowane.

Przykład 1

Przeanalizujmy przypadek, w którym tworzymy oprogramowanie sterujące silnikiem okrętu. Chcemy umożliwić sterowanie w taki sposób, aby przy ustawieniach: cała wstecz, silniki stop, cała naprzód niemożliwa była zmiana z pozycji cała naprzód do cała wstecz i odwrotnie – z pominięciem silniki stop. Dla celów wizualizacji użyjemy wypisywania danych na ekranie, z zastosowaniem techniki [f-string](#).

Napiszemy definicję klasy bez wykorzystania hermetyzacji.

```
1 class SilnikOkrętowy:
2     stany = ["Cała wstecz", "Silniki STOP", "Cała naprzód"]
3
4     def __init__(self):
5         self.wskazanie = 1
6         self.stan_silnika = self.stany[self.wskazanie]
7         self.aktualizuj_silniki(0)
8
9     def aktualizuj_silniki(self, krok):
10        if krok < 0 and self.wskazanie > 0:
11            print("Zmiana silników")
12            self.wskazanie += krok
13        if krok > 0 and self.wskazanie < 2:
14            print("Zmiana silników")
15            self.wskazanie += krok
16
17        self.stan_silnika = self.stany[self.wskazanie]
18        print(f"Aktualnie silniki: {self.stan_silnika}")
19
20    def silniki_naprzod(self):
21        self.aktualizuj_silniki(1)
22
23    def silniki_wstecz(self):
```

```

24         self.aktualizuj_silniki(-1)
25
26
27 # inicjalizacja zmiennej obiektowej
28 batory = SilnikOkretowy()
29 # Aktualnie silniki: Silniki STOP
30
31 batory.silniki_naprzod()
32 # Zmiana silników
33 # Aktualnie silniki: Cała naprzód
34
35 batory.silniki_naprzod()
36 # Aktualnie silniki: Cała naprzód
37
38 batory.silniki_naprzod()
39 # Aktualnie silniki: Cała naprzód
40
41 batory.silniki_wstecz()
42 # Zmiana silników
43 # Aktualnie silniki: Silniki STOP
44
45 batory.silniki_wstecz()
46 # Zmiana silników
47 # Aktualnie silniki: Cała wstecz
48
49 batory.silniki_wstecz()
50 # Aktualnie silniki: Cała wstecz

```

Definiowanie bez żadnych dodatkowych oznaczeń pól czy metod, które powinny być prywatne, może skończyć się błędami w kodzie. Np. ktoś nie wiedząc o tym, że nie powinien modyfikować jakiejś zmiennej, nadpisze jej wartość, co spowoduje błąd – jak w przedstawionym przykładzie:

```

1 print(batory.stan_silnika)
2 # Cała wstecz
3 print(batory.wskazanie)
4 # 0
5
6 batory.wskazanie = 2
7 print(batory.wskazanie)
8 # 2

```

```

9 print(batory.stan_silnika)
10 # Cała wstecz
11 batory.aktualizuj_silniki(0)
12 # Aktualnie silniki: Cała naprzód
13
14 # wskazanie jest zmienną która powinna być modyfikowana tylko
15 # przez przeznaczone do tego funkcje, ręczna zmiana jak tu może
16 # doprowadzić do tego, że będziemy się odwoływać do indeksu
17 # większego niż rozmiar tablicy i zostanie zwrócony błąd
18 batory.wskazanie = 5
19 batory.aktualizuj_silniki(0)
20 # Traceback (most recent call last):
21 #   File "<pyshell#55>", line 1, in <module>
22 #     batory.aktualizuj_silniki(0)
23 #   File "/home/python/idle-src/0373_silniki_okr_hermet.py", li
24 #     self.stan_silnika = self.stany[self.wskazanie]
25 # IndexError: list index out of range

```

Hermetyzacja w języku Python polega na umowie między programistami, że będą przestrzegać ogólnie przyjętych norm. Metody i pola, które chcemy oznaczyć jako prywatne (czyli takie, które nie powinny być używane poza daną klasą), oznaczamy znakiem "_" na początku nazwy zmiennej lub metody np. _x. Środowiska programistyczne często informują użytkownika, gdy w złym miejscu skorzysta ze zmiennej nazwanej w ten sposób.

```

1 class SilnikOkretowyPrywatny:
2     # zmiana tablicy na krotkę, ponieważ jest ona niezmienna (i
3     stany = ("Cała wstecz", "Silniki STOP", "Cała naprzód")
4
5     def __init__(self):
6         self._wskazanie = 1
7         self._stan_silnika = self.stany[self._wskazanie]
8         self._aktualizuj_silniki(0)
9
10    def _aktualizuj_silniki(self, krok):
11        if krok < 0 and self._wskazanie > 0:
12            print("Zmiana silników")
13            self._wskazanie += krok
14        if krok > 0 and self._wskazanie < 2:
15            print("Zmiana silników")
16            self._wskazanie += krok

```

```

17
18         self._stan_silnika = self.stany[self._wskazanie]
19         print(f"Aktualnie silniki: {self._stan_silnika}")
20
21     def silniki_naprzod(self):
22         self._aktualizuj_silniki(1)
23
24     def silniki_wstecz(self):
25         self._aktualizuj_silniki(-1)

```

Przy takim zapisie nadal możemy korzystać z metod, które nie są oznaczone jako prywatne.

```

1  pomerania = SilnikOkretowyPrywatny()
2  # Aktualnie silniki: Silniki STOP
3
4  pomerania.silniki_naprzod()
5  # Zmiana silników
6  # Aktualnie silniki: Cała naprzód
7
8  pomerania.silniki_naprzod()
9  # Aktualnie silniki: Cała naprzód
10
11 pomerania.silniki_naprzod()
12 # Aktualnie silniki: Cała naprzód
13
14 pomerania.silniki_wstecz()
15 # Zmiana silników
16 # Aktualnie silniki: Silniki STOP
17
18 pomerania.silniki_wstecz()
19 # Zmiana silników
20 # # Aktualnie silniki: Cała wstecz
21
22 pomerania.silniki_wstecz()
23 # Aktualnie silniki: Cała wstecz

```

Gdy spróbujemy użyć metod bądź pól ustawionych jako prywatne, okaże się, że wszystko działa bezproblemowo. Ostrzeżenie wysyłane przez środowisko to jedyny sygnał informujący nas, że coś jest nie w porządku.

```
1 pomerania._stan_silnika = "Silnik zniszczony"
2 print(pomerania._stan_silnika)
3 # Silnik zniszczony
```

Jeśli wykorzystamy zmienną lub metodę rozpoczynającą się od "_" (poza klasą, w której są zadeklarowane), możemy niecelowo doprowadzić do błędu działania programu.

Polimorfizm

Polimorfizm oznacza wielopostaciowość. Pozwala on zastosować jedną jednostkę (metodę, operator lub obiekt) do reprezentowania różnych typów w różnych scenariuszach.

Przykład 2

Przeanalizujmy przypadek, w którym tworzymy klasę `Statek`, opisującą ogólnie każdy obiekt pływający, a także klasy dziedziczące `Bryg` oraz `Fregata` (bryg oraz fregata to typy okrętów). Inicjalizację obiektów wykonamy za pomocą konstruktora zdefiniowanego w klasie `Statek`, wykorzystując funkcję `super()`. Pozwala to na używanie metod z klasy `Statek` w klasie potomnej. Dodatkowo przetestujemy działanie metod `info()` i `test()`. Dla celów wizualizacji użyjemy funkcji wypisywania danych na ekranie, z zastosowaniem techniki f-string.

```
1 class Statek:
2     def __init__(self, rok_wodowania):
3         self._rok_wodowania = rok_wodowania
4         print(f"Utworzono obiekt {self}.")
5
6     def nowy_rok(self):
7         self._rok_wodowania += 1
8
9     def rok_wodowania(self):
10        return self._rok_wodowania
11
12 class Bryg(Statek):
13     def __init__(self, rok_wodowania):
14         super().__init__(rok_wodowania)
15         self._typ = "Bryg/2 maszty"
16
17     def info(self):
18         print(f"Metoda w klasie {__class__.__name__}")
19         print(f"Obiekt {self} - typ: {self._typ}")
```

```

20         print(f"Rok wodowania = {super().rok_wodowania()}")
21
22     def test(self):
23         print("To jest wywołanie test-Bryg (rok_wodowania + 20)")
24         print(f"wynik = {super().rok_wodowania()+20}")
25
26 class Fregata(Statek):
27     def __init__(self, rok_wodowania):
28         super().__init__(rok_wodowania)
29         self._typ = "Fregata/3 maszty"
30
31     def info(self):
32         print(f"Metoda w klasie {__class__.__name__}")
33         print(f"Obiekt {self} - typ: {self._typ}")
34         print(f"Rok wodowania = {super().rok_wodowania()}")
35
36     def test(self):
37         print("To jest wywołanie test-Fregata (rok_wodowania + 20)")
38         print(f"wynik = {super().rok_wodowania()+40}")
39
40
41 # definiujemy obiekty oraz tworzymy listę --- wartości ID obiektów
42 statek01 = Fregata(2001)
43 statek02 = Fregata(2002)
44 statek03 = Bryg(2010)
45 statek04 = Bryg(2012)
46 sts = Bryg(1992)
47 cutty_sark = Fregata(1869)
48
49 # Utworzono obiekt <__main__.Fregata object at 0x7f22a2ffac50>.
50 # Utworzono obiekt <__main__.Fregata object at 0x7f22a410dba8>.
51 # Utworzono obiekt <__main__.Bryg object at 0x7f22a410dc18>.
52 # Utworzono obiekt <__main__.Bryg object at 0x7f22a2ffadd8>.
53 # Utworzono obiekt <__main__.Bryg object at 0x7f22a2ffada0>.
54 # Utworzono obiekt <__main__.Fregata object at 0x7f22a2ffad30>.
55
56 spis_statkow = [statek01, sts, statek02, statek03, cutty_sark,

```

Możemy teraz napisać kod, który sprawdzi, czy obiekty są uznawane za instancje klasy Statek. W języku Python sprawdzenie przynależności danego obiektu do klasy wykonuje się metodą `isinstance()`.

```

1 for statek in spis_statkow:
2     print(f"isinstance({statek}, Bryg) = {isinstance(statek, Br
3     print(f"isinstance({statek}, Fregata) = {isinstance(statek,
4     print(f"isinstance({statek}, Statek) = {isinstance(statek,
5     print("-----")
6
7
8 # isinstance(<__main__.Fregata object at 0x7f22a2ffac50>, Bryg)
9 # isinstance(<__main__.Fregata object at 0x7f22a2ffac50>, Frega
10 # isinstance(<__main__.Fregata object at 0x7f22a2ffac50>, State
11 # -----
12 # isinstance(<__main__.Bryg object at 0x7f22a2ffada0>, Bryg) =
13 # isinstance(<__main__.Bryg object at 0x7f22a2ffada0>, Fregata)
14 # isinstance(<__main__.Bryg object at 0x7f22a2ffada0>, Statek)
15 # -----
16 # isinstance(<__main__.Fregata object at 0x7f22a410dba8>, Bryg)
17 # isinstance(<__main__.Fregata object at 0x7f22a410dba8>, Frega
18 # isinstance(<__main__.Fregata object at 0x7f22a410dba8>, State
19 # -----
20 # isinstance(<__main__.Bryg object at 0x7f22a410dc18>, Bryg) =
21 # isinstance(<__main__.Bryg object at 0x7f22a410dc18>, Fregata)
22 # isinstance(<__main__.Bryg object at 0x7f22a410dc18>, Statek)
23 # -----
24 # isinstance(<__main__.Fregata object at 0x7f22a2ffad30>, Bryg)
25 # isinstance(<__main__.Fregata object at 0x7f22a2ffad30>, Frega
26 # isinstance(<__main__.Fregata object at 0x7f22a2ffad30>, State
27 # -----
28 # isinstance(<__main__.Bryg object at 0x7f22a2ffadd8>, Bryg) =
29 # isinstance(<__main__.Bryg object at 0x7f22a2ffadd8>, Fregata)
30 # isinstance(<__main__.Bryg object at 0x7f22a2ffadd8>, Statek)

```

Możemy również pokazać polimorfizm – w tym celu, dla wszystkich obiektów z listy `spis_statkow`, dziedziczących po klasie `Statek`, wywołamy metodę `test()`. Zauważ, że metoda działa inaczej dla obiektów różnych klas.

```

1 for statek in spis_statkow:
2     print("-----")
3     statek.test()
4     print("----")
5     statek.info()

```

```
6
7
8 # -----
9 # To jest wywołanie test-Fregata (rok_wodowania + 40)
10 # wynik = 2041
11 # -----
12 # Metoda w klasie Fregata
13 # Obiekt <__main__.Fregata object at 0x7f22a2ffac50> - typ: Fre
14 # Rok wodowania = 2001
15 # -----
16 # To jest wywołanie test-Bryg (rok_wodowania + 20)
17 # wynik = 2012
18 # -----
19 # Metoda w klasie Bryg
20 # Obiekt <__main__.Bryg object at 0x7f22a2ffada0> - typ: Bryg/2
21 # Rok wodowania = 1992
22 # -----
23 # To jest wywołanie test-Fregata (rok_wodowania + 40)
24 # wynik = 2042
25 # -----
26 # Metoda w klasie Fregata
27 # Obiekt <__main__.Fregata object at 0x7f22a410dba8> - typ: Fre
28 # Rok wodowania = 2002
29 # -----
30 # To jest wywołanie test-Bryg (rok_wodowania + 20)
31 # wynik = 2030
32 # -----
33 # Metoda w klasie Bryg
34 # Obiekt <__main__.Bryg object at 0x7f22a410dc18> - typ: Bryg/2
35 # Rok wodowania = 2010
36 # -----
37 # To jest wywołanie test-Fregata (rok_wodowania + 40)
38 # wynik = 1909
39 # -----
40 # Metoda w klasie Fregata
41 # Obiekt <__main__.Fregata object at 0x7f22a2ffad30> - typ: Fre
42 # Rok wodowania = 1869
43 # -----
44 # To jest wywołanie test-Bryg (rok_wodowania + 20)
45 # wynik = 2032
46 # -----
47 # Metoda w klasie Bryg
```

```
48 # Obiekt <__main__.Bryg object at 0x7f22a2ffadd8> - typ: Bryg/2
49 # Rok wodowania = 2012
```

Dodatkowo możemy wykonać metodę, która jest zdefiniowana w klasie Statek, a wynik jej działania zobaczymy podczas uruchamiania kolejnej metody.

```
1 for statek in spis_statkow:
2     statek.nowy_rok()
3 # tu nie ma żadnej informacji zwrotnej, metoda tylko aktualizuj
4
5 for statek in spis_statkow:
6     statek.info()
7
8
9 # Metoda w klasie Fregata
10 # Obiekt <__main__.Fregata object at 0x7f22a2ffac50> - typ: Fre
11 # Rok wodowania = 2002
12 # Metoda w klasie Bryg
13 # Obiekt <__main__.Bryg object at 0x7f22a2ffada0> - typ: Bryg/2
14 # Rok wodowania = 1993
15 # Metoda w klasie Fregata
16 # Obiekt <__main__.Fregata object at 0x7f22a410dba8> - typ: Fre
17 # Rok wodowania = 2003
18 # Metoda w klasie Bryg
19 # Obiekt <__main__.Bryg object at 0x7f22a410dc18> - typ: Bryg/2
20 # Rok wodowania = 2011
21 # Metoda w klasie Fregata
22 # Obiekt <__main__.Fregata object at 0x7f22a2ffad30> - typ: Fre
23 # Rok wodowania = 1870
24 # Metoda w klasie Bryg
25 # Obiekt <__main__.Bryg object at 0x7f22a2ffadd8> - typ: Bryg/2
26 # Rok wodowania = 2013
```

Dla zainteresowanych

W języku Python przykładem polimorfizmu jest operator „+”. Działa on w różny sposób w zależności od typu danych.

```
1 # przykład działania operatora + dla typów numerycznych
2 print(3 + 3)
```

```
3 # 6
4 print(3 + 3.0)
5 # 6.0
6 # przykład działania operatora + dla typów znakowych
7 print("3" + "3")
8 # 33
```

Dla zainteresowanych

Innym przykładem polimorfizmu w języku Python jest funkcja `len()`, która działa dla różnych typów danych w ten sam sposób.

```
1 # przykład działania
2 # str
3 print(len("Python to język programowania."))
4 # 30
5 # dict
6 print(len({1: "Python", 2: "Linux", 3: "Open Source"}))
7 # 3
8 # list
9 print(len([1, 3, 4, [1, 1], "Python", True, 3.14]))
10 # 7
11 # tuple
12 print(len((1, 3, 4, "Python", True, 3.14)))
13 # 6
14 # niektóre typy danych nie są obsługiwane przez funkcję len()
15 # int
16 print(len(4))
17 # Traceback (most recent call last):
18 #   File "d:\Programming\VSCProjects\Poprawianie\0431\test.py",
19 #     print(len(4))
20 # TypeError: object of type 'int' has no len()
```

Już wiesz

Podsumujmy najważniejsze elementy tego e-materiału:

- język programowania Python jest w pełni językiem obiekowym,
- zawiera narzędzia i sposoby na bezproblemowe stosowanie obiekowego paradygmatu programowania,
- pozwala wygodnie tworzyć klasy, obiekty, właściwości i metody.

Słownik

f-string

sposób zapisywania zmiennych w łańcuchu znaków przeznaczonym do wyświetlenia za pomocą funkcji `print()`; dostępny od wersji 3.6 języka Python; polecenie wykorzystujące mechanizm f-string ma postać:

`f"Napis, a w nim {zmienna} do wypisania"`; dokładnie opisany w dokumencie PEP 498 – *Literal String Interpolation*

Prezentacja multimedialna

Polecenie 1

Zapoznaj się z prezentacją. Przeanalizuj zachowanie zasad poliformizmu na przykładzie metody `maszty_opis()` i hermetyzacji na przykładzie pól `_maszty` i `_typ`.

Materiał audio dostępny pod adresem:

<https://zpe.gov.pl/b/PJGzxgfR4>

1

Użyjemy definicji klasy `Statek`, która w konstruktorze przyjmuje i ustawia wartości pól `_typ` i `_maszty`. Jeśli `typ` nie zawiera się w zakresie $\langle 0, 3 \rangle$ to jest ustawiany na 4, co oznacza, że jest to nieznany typ statku. Klasa posiada także metody `typ_opis()`, `info()` i `maszty_opis()`, które są odpowiedzialne za wyświetlanie informacji o statku.

2

Materiał audio dostępny pod adresem:

<https://zpe.gov.pl/b/PJGzxgfR4>

Tworzymy obiekt `s1` klasy `Statek`. Obiekt ten ma 0 masztów i reprezentuje typ `Tylko kadłub`:

Jest tak, ponieważ przyjmuje wartości domyślne, gdyż nie ustawiliśmy żadnych wartości pól dla tego obiektu klasy `Statek`.

3

Materiał audio dostępny pod adresem:

<https://zpe.gov.pl/b/PJGzxgfR4>

Tworzymy obiekt s2 klasy Statek mający 2 maszty i będący typu Bryg:

|

4

Materiał audio dostępny pod adresem:

<https://zpe.gov.pl/b/PJGzxgfR4>

Tworzymy obiekt s3 klasy Statek mający 0 masztów i będący typu Nieznany (ponieważ 5 jest większe niż 3):

|

5

Materiał audio dostępny pod adresem:

<https://zpe.gov.pl/b/PJGzxgfR4>

Tworzymy obiekt s4 klasy Statek, którego wartość pola _maszty wynosi Brak, a jego typ to Klipper:

|

Zwróć uwagę, że typ nie jest wyrażony liczbą.

6

Materiał audio dostępny pod adresem:

<https://zpe.gov.pl/b/PJGzxgfR4>

Sprawdzamy informacje dostępne dla obiektów:

Materiał audio dostępny pod adresem:

<https://zpe.gov.pl/b/PJGzxgfR4>

7

Sprawdzamy działanie metody
`maszty_opis()` dla obiektów:

8

Materiał audio dostępny pod adresem:

<https://zpe.gov.pl/b/PJGzxgfR4>

W przedstawionym kodzie zastosowaliśmy
hermetyzację i polimorfizm, które są
podstawowymi założeniami obiektowego
paradygmatu programowania. Cały kod
programu:

Źródło: Contentplus.pl Sp. z o.o., licencja: CC BY-SA 3.0.

Polecenie 2

Przygotuj notatkę, w której podsumujesz najważniejsze informacje zawarte w prezentacji

Sprawdź się

Pokaż ćwiczenia:   

Ćwiczenie 1



Przeanalizuj przedstawiony kod.

```
1 class Counter:
2     def __init__(self, count):
3         # miejsce na definicję pola instancji klasy
4
5     def t(self):
6         print(f"Wartość => {self."tu wprowadź nazwę zdefiniowa
7
8
9 # wykonanie metody
10 c = Counter(5)
11 c.t()
```

Wskaż, która definicja pola instancji klasy jest poprawna, gdy chcemy oznaczyć ją jako zmienną, która nie powinna być modyfikowana poza klasą.

☐ self._count = count

☐ self.__count = count

☐ self.count = count

Ćwiczenie 2



Zdefiniuj dwie klasy: pierwszą o nazwie `Punkt` oraz dziedziczącą z niej klasę `KolorowyPunkt`.

Niech w klasie istnieje konstruktor przypisujący wartość do pól o nazwie `x` i `y` oraz metoda o nazwie `wypisz_wspolrzedne()`, wypisująca f-string: `"Położenie punktu to ({x}, {y})"`.

Niech w klasie `KolorowyPunkt` istnieje korzystający z konstruktora klasy `Punkt` (`super`), który dodatkowo ustawi wartość pola `kolor`. Jego zadaniem jest sprawdzenie, czy wartość jest typu `string`. W przeciwnym wypadku powinien przypisać domyślną wartość `"czerwony"` do pola `kolor`. Dla klasy zdefiniuj również metodę o nazwie `wypisz_kolor()`, wypisująca f-string: `"Kolor punktu to {kolor}"`.

Specyfikacja problemu:

Dane:

Pole instancji klasy A:

- `x` – liczba całkowita
- `y` – liczba całkowita

Pole instancji klasy B:

- `x` – liczba całkowita
- `y` – liczba całkowita
- `kolor` – łańcuch znaków

Wynik:

Program, na standardowe wyjście, wypisuje wartości przez wywołanie metody `wypisz_wspolrzedne()` dla obiektu klasy `Punkt` i metod `wypisz_wspolrzedne()`, `wypisz_kolor()` dla obiektów klasy `KolorowyPunkt`.

Przetestuj program dla obiektu klasy `Punkt` z wartością pola `x` równą 2 i pola `y` równą 1, obiektu klasy `KolorowyPunkt` z wartością pól: `x = 3`, `y = 7`, `kolor = -5` i obiektu klasy `B` z wartością pól: `x = 8`, `y = -3`, `kolor = „niebieski”`.

Twoje zadania

1. Program wypisuje wartości zwrócone przez wywołanie metod `wypisz_wspolrzedne()` i `wypisz_kolor()` klas `Punkt` i `KolorowyPunkt`.

```
1 class Punkt:
2     pass
3
4
5 class KolorowyPunkt(Punkt):
6     pass
7
8
9 punkt1 = Punkt(2, 1)
10 punkt2 = KolorowyPunkt(3, 7, -5)
11 punkt3 = KolorowyPunkt(8, -3, "niebieski")
12
13 punkt1.wypisz_wspolrzedne()
14 punkt2.wypisz_wspolrzedne()
```

```
1
```





Korzystając z przedstawionego kodu, utwórz klasę `Policjant`, która będzie dziedziczyć po klasie `Pracownik`. W klasie `Policjant` zadeklaruj konstruktor, który będzie przyjmował dwa parametry typu `String`: `imie` oraz `nazwisko`. Zadbaj, aby pole `miejscePracy` w klasie `Policjant` było ustawione na komisariat. Następnie, przez nadpisanie metody `przedstawSie()`, popraw błąd językowy tak, aby program wypisywał komunikat `Nazywam się {imie} {nazwisko}. Pracuję na komisariacie.` Nie modyfikuj istniejącej klasy `Pracownik` oraz funkcji głównej programu. Program przetestuj dla imienia Jan i nazwiska Kowalski.

Specyfikacja problemu:

Dane:

- `imie` – imię; ciąg znaków
- `nazwisko` – nazwisko; ciąg znaków

Wynik:

Program, na standardowe wyjście, wypisuje zadany komunikat.

Twoje zadania

1. Program wypisuje komunikat z imieniem i nazwiskiem policjanta, a także jego miejsce pracy

```
1 class Pracownik:
2     def __init__(self, imie: str, nazwisko: str,
3     miejsce_pracy: str):
4         self.imie = imie
5         self.nazwisko = nazwisko
6         self.miejsce_pracy = miejsce_pracy
7
8     def przedstaw_sie(self):
9         print(f"Nazywam się {self.imie} {self.nazwisko}.
10        Pracuję w {self.miejsce_pracy}")
```

```
9
10 class Policjant(Pracownik):
11     pass # Tu wpisz swój kod
12
```

```
1
```

Dla nauczyciela

Autor: Adam Jurkiewicz

Przedmiot: Informatyka

Temat: Zasady programowania obiektowego w języku Python

Grupa docelowa:

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres rozszerzony

Podstawa programowa:

Cele kształcenia – wymagania ogólne

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera oraz innych urządzeń cyfrowych: układanie i programowanie algorytmów, organizowanie, wyszukiwanie i udostępnianie informacji, posługiwanie się aplikacjami komputerowymi.

Treści nauczania – wymagania szczegółowe

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera i innych urządzeń cyfrowych.

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

2) stosuje zasady programowania strukturalnego i obiektowego w rozwiązywaniu problemów;

Kształtowane kompetencje kluczowe:

- kompetencje cyfrowe;
- kompetencje osobiste, społeczne i w zakresie umiejętności uczenia się;
- kompetencje matematyczne oraz kompetencje w zakresie nauk przyrodniczych, technologii i inżynierii.

Cele operacyjne (językiem ucznia):

- Prześledzisz zaawansowane zasady programowania obiektowego w języku Python.
- Zaimplementujesz definicje klas wykorzystujących techniki programowania obiektowego.
- Stworzysz program wykorzystujący techniki programowania obiektowego i definicje klas.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimediu i ćwiczeń interaktywnych;
- ćwiczenia praktyczne.

Formy pracy:

- praca indywidualna;
- praca w parach;
- praca w grupach;
- praca całego zespołu klasowego.

Środki dydaktyczne:

- komputery z głośnikami, słuchawkami i dostępem do internetu;
- zasoby multimedialne zawarte w e-materiale;
- tablica interaktywna/tablica, pisak/kreda;
- oprogramowanie dla języka Python 3 (lub nowszej wersji), w tym PyCharm lub IDLE.

Przebieg lekcji

Przed lekcją:

1. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Zasady programowania obiektowego w języku Python”. Uczniowie zapoznają się z treściami w sekcji „Przeczytaj” w kontekście programowania.

Faza wstępna:

1. Nauczyciel wyświetla i odczytuje temat lekcji oraz cele zajęć. Prosi uczniów o sformułowanie kryteriów sukcesu.
2. Prowadzący prosi uczniów, aby zgłaszali swoje propozycje pytań do tematu. Jedna osoba może zapisywać je na tablicy. Gdy uczniowie wyczerpią swoje pomysły, a pozostały jakieś ważne kwestie do poruszenia, nauczyciel je dopowiada.

Faza realizacyjna:

1. Nauczyciel wyświetla zawartość sekcji „Przeczytaj”. Na forum klasy uczniowie analizują przedstawione w niej rozwiązania Przykładów 1 i 2. A następnie testują je na swoich komputerach.

2. **Praca z multimediami.** Nauczyciel wyświetla zawartość sekcji „Prezentacja multimedialna”. Uczniowie zapoznają się indywidualnie z treścią prezentacji, analizują zachowanie zasad polimorfizmu na przykładzie metody `maszty_opis()` i hermetyzacji na przykładzie pól `_maszty` i `_typ`. Nauczyciel wyjaśnia ewentualne niezrozumiałe kwestie.
3. **Ćwiczenie umiejętności.** Prowadzący zapowiada uczniom, że będą rozwiązywać ćwiczenie nr 1 z sekcji „Sprawdź się”. Każdy z uczniów robi to samodzielnie. Po ustalonym czasie następuje porównanie napisanych kodów podczas wspólnego omówienia rozwiązań.
4. Uczniowie w parach wykonują ćwiczenia nr 2. Nauczyciel sprawdza poprawność wykonanych zadań, omawiając je wraz z uczniami.

Faza podsumowująca:

1. Wybrany uczeń podsumowuje zajęcia, zwracając uwagę na nabyte umiejętności, omawia ewentualne problemy podczas rozwiązywania ćwiczeń z programowania w języku Python.

Praca domowa:

1. Uczniowie wykonują ćwiczenie nr 3 z sekcji „Sprawdź się”.

Materiały pomocnicze:

- Oficjalna dokumentacja techniczna dla języka Python 3 (lub nowszej wersji).
- Oficjalna dokumentacja techniczna dla oprogramowania PyCharm lub IDLE.

Wskazówki metodyczne:

- Treści w sekcji „Przeczytaj” można wykorzystać jako podsumowanie i utrwalenie wiedzy uczniów.