Python

Poziom średniozaawansowany

Cześć!

Łukasz Paluch

Technical Team Lead w IG KnowHow

Kontakt: ap.lukaszpaluch@gmail.com

Garść informacji organizacyjnych

1. Jak wyglądać będzie szkolenie?

Garść informacji organizacyjnych

1. Jak wyglądać będzie szkolenie?
2. Co z przerwami?

Garść informacji organizacyjnych

1. Jak wyglądać będzie szkolenie?
2. Co z przerwami?
3. Jak wam się podoba Python???

Garść informacji organizacyjnych

1. Jak wyglądać będzie szkolenie?
2. Co z przerwami?
3. Jak wam się podoba Python???
4. Jakie tematy będą poruszane w trakcie tego bloku?

Plan

1. Python – ćwiczenia

- zmienne, typy zmiennych, instrukcje warunkowe, pętle, funkcje
- programowanie obiektowe + zagadnienia zaawansowane z OOP

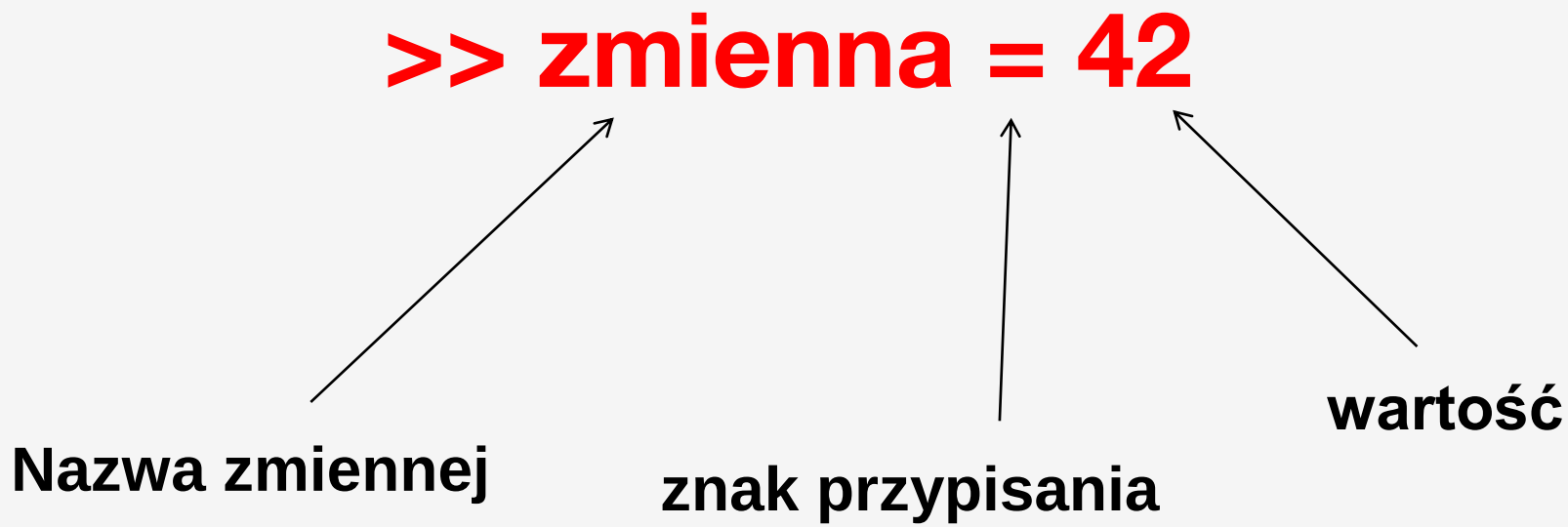
2. Python średniozaawansowany

- funkcje – zagadnienia zaawansowane
- wyrażenia lambda
- dekoratory
- generatory
- operacje na plikach
- wyjątki
- wyrażenia regularne

ZACZYNAMY!

Python ćwiczenia – zmienne i typy

Zmienna -> pojemnik na dane



Python ćwiczenia – zmienne i typy

Każda zmienna posiada:

- nazwę
- wartość
- typ*
- miejsce w pamięci

```
In [1]: zmienna = 42

In [2]: type(zmienna)
Out[2]: int
```

```
In [1]: zmienna = 42

In [2]: id(zmienna)
Out[2]: 1352980160

In [3]: hex(id(zmienna))
Out[3]: '0x50a4d6c0'
```

Python ćwiczenia – zmienne i typy

Istnieje pokaźna ilość typów danych (typy warunkują ilość pamięci zajmowanej przez zmienną):

- Liczby (int – liczby całkowite, float – liczby wymierne (zmiennoprzecinkowe))
- ❖ **1, -44, 100, 0, 5.5, 3.1415**
- Napisy (w informatyce nazywane stringami)
- ❖ **''Hello world'', 'napis', '''wielonijkowy napis'''**
- Wartości boolowskie (prawda/fałsz)
- ❖ **True, False**
- Listy i krotki/tuple (zbiór zmiennych różnych typów)
- ❖ **[55, 'napis', True]** -> lista, **(55, 'napis', True)** -> krotka/tuple
- Słowniki (mapa klucz -> wartość)
- ❖ **{'Krzysztof': 4, 'Halina': 4.5, 'Wojciech': 5}**
- Zbiory (kolekcja niepowtarzających się danych)
- ❖ **{1,2,3,4}** – wartości nie mogą się powtarzać
- Reprezentacja niczego: **None**

Python ćwiczenia – zmienne i typy

Uruchommy terminal i sprawdźmy czy rozumiemy co dzieje się w poszczególnych instrukcjach:

1. 2 ** 16
2. 2 / 5, 2 / 5.0
3. "abc" + "123"
4. s = "napis"
5. "napis " + s
6. s * 5
7. s[:0]
8. "zielony %s i %s" % ("napis", s)
9. 'zielony {0} i {1}'.format(‘napis', s)
10. f'zielony {‘napis', } i {s}'

Python ćwiczenia – zmienne i typy

1. ('x',)[0]
2. ('x', 'y')[1]
3. L = [1,2,3] + [4,5,6]
4. L, L[:], L[:0], L[-2], L[-2:]
5. ([1,2,3] + [4,5,6])[2:4]
6. [L[2], L[3]]
7. L.reverse(); L
8. L.sort(); L
9. L.index(4)

Python ćwiczenia – zmienne i typy

1. {'a':1, 'b':2}['b']
2. D = {'x':1, 'y':2, 'z':3}
3. D['w'] = 0
4. D['x'] + D['w']
5. D[(1,2,3)] = 4
6. list(D.keys()), list(D.values()), (1,2,3) in D
7. [[]]
8. ["",[],(),{},None]

Python ćwiczenia – zmienne i typy

W sesji interaktywnej stwórz czteroelementową listę o nazwie L:

• Co się stanie, kiedy spróbujemy wykorzystać indeks znajdujący się poza długością listy (jak L[4])?

• Co się stanie, kiedy spróbujemy wykonać wycinek wykraczający poza długość listy (jak L[-100:100])?

• Jak radzi sobie Python, kiedy próbujemy dokonać ekstrakcji sekwencji w odwrotnej kolejności — gdy niższa granica jest większa od wyższej (jak L[3:1])? (Wskazówka: warto spróbować przypisać coś do tego wycinka (na przykład L[3:1] = ['?']) i zobaczyć, gdzie zostanie wstawiona wartość.

Python ćwiczenia – zmienne i typy

Utwórz strukturę danych reprezentującą dane osobowe. Zawrzyj w tej strukturze takie informacje jak imiona i nazwisko, wiek, zawód, adres, e-mail i numer telefonu. Wykorzystaj do tego dowolnie wybrane zmienne (przeróżnych typów). Spróbuj potem uzyskać dostęp do poszczególnych elementów za pomocą indeksowania.

Python ćwiczenia – instrukcje warunkowe

Jeżeli program ma podjąć jedno z kilku działań, wykorzystuje do tego instrukcję **if**.

```
if <warunek>:
    <instrukcje>
elif <inny_warunek>:
    <inne_instrukcje>
else:
    <jeszcze_inne_instrukcje>
```

Python ćwiczenia – instrukcje warunkowe

Python musi być w stanie przekształcić **warunek** w wyrażenie boolowskie, czyli takie, których wynik przyjąć może jedną z dwóch wartości: <True, False>.

```
In [15]: True and True
Out[15]: True

In [16]: True and False
Out[16]: False

In [17]: False or True
Out[17]: True

In [18]: not False
Out[18]: True

In [19]: not True
Out[19]: False
```

```
In [20]: "a" == "b"
Out[20]: False

In [21]: 10 is 10
Out[21]: True

In [22]: "abc" > "Abc"
Out[22]: True

In [23]: 22 != 30
Out[23]: True
```

```
In [30]: bool(100)
Out[30]: True

In [31]: bool(0)
Out[31]: False

In [32]: bool("napis")
Out[32]: True

In [33]: bool("")
Out[33]: False

In [34]: bool([])
Out[34]: False

In [35]: bool({}) == False
Out[35]: True
```

Python ćwiczenia – instrukcje warunkowe

Napisz skrypt pytający użytkownika o wiek. Jeżeli użytkownik jest przed 18-tką wyświetli informację „Użytkownik niepełnoletni” oraz zwróci ile lat zostało użytkownikowi do pełnoletności. Użytkownikom pełnoletnim wyświetli informację „Użytkownik pełnoletni”. Sprawdź czy wiek użytkownika nie przekracza 100 lat i wyświetl komunikat „200 lat!!!”.

Python ćwiczenia – instrukcje warunkowe

Napisz program, który:

- wypisze wszystkie liczby od 1 do 100
- jeśli liczba jest podzielna przez trzy wypisze “**Fizz**“ zamiast liczby.
- jeśli liczba jest podzielna przez pięć wypisze “**Buzz**“ zamiast liczby.
- jeśli liczba jest podzielna przez trzy i pięć wypisze “**FizzBuzz**“ zamiast liczby.

Python ćwiczenia – instrukcje warunkowe

Napisz program, który znajdzie liczby między 2000 a 3200, które są podzielne przez 7, ale niepodzielne przez 5 i wyświetli je na ekranie.

Python ćwiczenia - pętle

W Pythonie występują dwa rodzaje pętli:

- Pętla for – stosujemy kiedy wiemy ile razy pętla ma się wykonać
- Pętla while – stosujemy kiedy nie wiemy ile razy pętla ma się wykonać i czekamy na spełnienie pewnego warunku

```
In [4]: for i in range(10):
   ...:     print(i)
   ...:
0
1
2
3
4
5
6
7
8
9
```

```
In [5]: i = 10

In [6]: while i > 7:
   ...:     print(f"i={i} jest wieksze od 7")
   ...:     i-=1
   ...:
i=10 jest wieksze od 7
i=9 jest wieksze od 7
i=8 jest wieksze od 7
```

Python ćwiczenia - pętle

W pętlach stosuje się dwie istotne komendy:

- continue – kiedy chcemy (nagle) rozpocząć kolejną iterację
- break – kiedy chcemy (nagle) przerwać pętlę zanim zakończy swoje działanie

```
In [7]: liczby = [12, 15, 5, -6]

In [8]: for liczba in liczby:
   ...:     print(liczba)
   ...:     if liczba == 15:
   ...:         break
   ...:
12
15
```

```
In [9]: liczby = [1, 2, 3, 4, 5]

In [10]: for liczba in liczby:
    ...:     if liczba != 4:
    ...:         print(liczba)
    ...:         continue
    ...:     break
    ...:
1
2
3
```

Analogicznie przykłady wyglądają z wykorzystaniem pętli while

Python ćwiczenia - pętle

W komputerach wszystko zapisywane jest jako liczba (komputer zrozumie np taki ciąg zer i jedynek: 01010110110110), nawet litery są przechowywane w pamięci jako liczby. Każdemu znakowi, od A-Z i a-z przypisywana jest konkretna liczba, standard ten nosi nazwę standardu ASCII.

Napisz pętlę **for** wyświetlającą kod ASCII dla każdego znaku z łańcucha o nazwie S. Do konwersji każdego znaku na kod ASCII należy wykorzystać wbudowaną funkcję **ord(*znak)***. Przetestuj jak działa ta funkcja w terminalu Pythona.

```
>> ord(‘A’)
>> 65
```

Python ćwiczenia - pętle

Utwórz pętlę, która będzie wyświetlała poniższy wzór:

```
*
**
***
****
*****
```

Python ćwiczenia - pętle

Napisz prosty program, który w każdym obiegu pętli zapyta użytkownika o liczbę i wypiszę ją na ekranie. Program ma się wykonywać do momentu, kiedy użytkownik poda liczbę podzielną przez 12.

Python ćwiczenia - pętle

Napisz program proszący użytkownika o podanie dwóch liczb całkowitych. Program powinien podać sumę wszystkich liczb całkowitych znajdujących się między podanymi liczbami (z nimi włącznie).

Jeżeli użytkownik poda liczby 4 i 8, program powinien wypisać 30, ponieważ 4+5+6+7+8=30.

Python ćwiczenia - funkcje

Funkcje to bloki, z których zbudowany jest program. Dzięki stosowaniu funkcji można używać raz napisany kod w wielu miejscach/plikach/pakietach bez wprowadzania powtarzalności. Dzięki temu w przypadku konieczności aktualizacji funkcji, wystarczy to zrobić w jednym miejscu.

```
def function(arg1, arg2):
    <instrukcje>
    return <wartość/wyrażenie>
```

Słowo kluczowe
do tworzenia funkcji

Nazwa funkcji

Lista argumentów przekazywanych do funkcji

Słowo kluczowe, dzięki któremu funkcja może zwrócić jakąś obliczoną wartość

Python ćwiczenia - funkcje

Ciekawostka: funkcja zawsze zwraca jakąś wartość, nawet jeśli pominiemy w jej ciele słowo return!

```
In [36]: def moja_funkcja():
    ...:     print("udaje, ze nic nie zwracam")
    ...:

In [37]: wartosc = moja_funkcja()
udaje, ze nic nie zwracam

In [38]: wartosc

In [39]: wartosc is None
Out[39]: True
```

W przypadku pominięcia słowa return, funkcja zwróci None.

Python ćwiczenia - funkcje

Zaimplementuj własną funkcję znajdującą maksymalną liczbę w liście liczb podanej jako parametr wejściowy.

Python ćwiczenia - funkcje

Napisz funkcję obliczającą silnię liczby podanej jako argument.

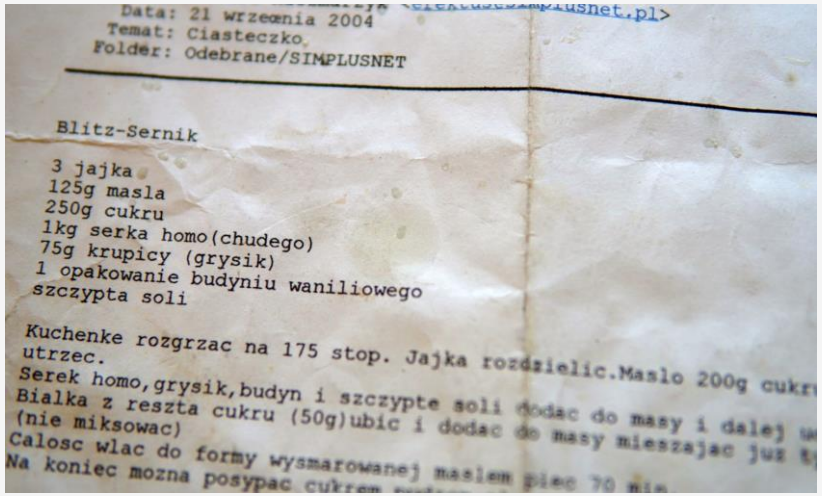
Python ćwiczenia – programowanie obiektowe

Po co nam programowanie obiektowe?

- Ponowne wykorzystanie kodu,
- Hermetyzacja
- Struktura
- Utrzymywanie

Zamysł: dane > algorytmy, dane są ważniejsze od zadań/instrukcji.

Klasa vs obiekt



Data: 21 wrzeœnia 2004
Temat: Ciasteczko
Folder: Odebrane/SIMPLUSNET

Blitz-Sernik

3 jajka
125g masla
250g cukru
1kg serka homo(chudego)
75g krupicy (grysik)
1 opakowanie budyniu waniliowego
szczypta soli

Kuchenke rozgrzac na 175 stop. Jajka rozdzielic.Maslo 200g cukru utrzec.
Serek homo,grysik,budyn i szczypte soli dodac do masy i dalej
Bialka z reszta cukru (50g)ubic i dodac do masy mieszajac juz
(nie miksowac)
Calosc wlac do formy wysmarowanej maslem piec 70 min
Na koniec mozna posypac cukrem

Klasa to nowy typ danych (zdefiniowany przez użytkownika), jest to przepis na utworzenie obiektu.

Python ćwiczenia – programowanie obiektowe

Definicja najprostszej klasy wygląda tak:

```
In [2]: class Sernik:
   ...:     pass
   ...:
```

W tym momencie powołaliśmy do życia nowy typ (obok int, float, str, list, itd.)

Python ćwiczenia – programowanie obiektowe

Analogia do podstawowych typów danych:

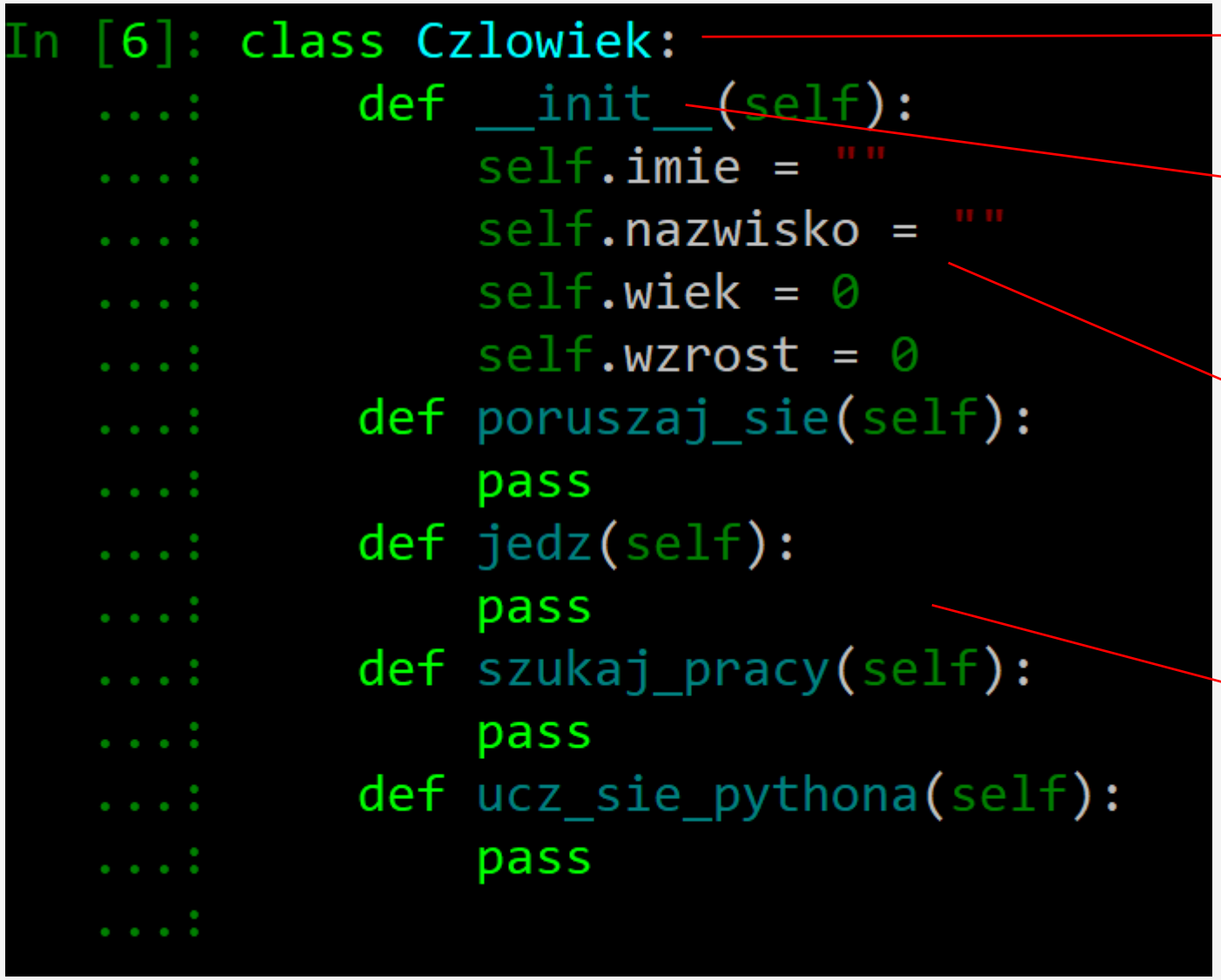
```
In [1]: type(10)
Out[1]: int
```

```
In [2]: class Sernik:
   ...:     pass
   ...:

In [3]: sernik_babci_kunegundy = Sernik()

In [4]: type(sernik_babci_kunegundy)
Out[4]: __main__.Sernik
```

Można by rzec, iż int to nazwa klasy, podczas gdy 10 to obiekt typu int. Podobnie Sernik to nazwa nowego typu, a sernik_babci_kunegundy to zmienna reprezentująca ten typ.

W związku z tym, że obiekt zazwyczaj reprezentuje coś realnego, coś co istnieje poza ekranem monitora, musi posiadać w sobie elementy, które w sposób oczywisty zidentyfikują go jako reprezentację realnego przedmiotu: musi on mieć podobne cechy i podobnie się zachowywać.

Python ćwiczenia – programowanie obiektowe



Definicja klasy Czlowiek, utworzenie nowego typu

Konstruktor – specjalna funkcja odpowiedzialna za definiowanie atrybutów przyszłego obiektu, inicjalizowanie wartości tych atrybutów oraz tworzenie samego obiektu

Deklaracja atrybutów, które będzie posiadał każdy obiekt typu Czlowiek

Definicja metod, czyli akcji, jakie będzie można wykonywać na obiekcie

OBIEKT, KLASA, POLA, METODY - ĆWICZENIE

Python ćwiczenia – programowanie obiektowe

Paradygmaty programowania obiektowego:

- ***Abstrakcja*** – skupienie się na wydobyciu informacji niezmiennych i wspólnych dla pewnej grupy obiektów.
- ***Enkapsulacja*** – ukrywanie szczegółów implementacyjnych przed użytkownikiem.
- ***Dziedziczenie*** – sposób na rozszerzanie klas na podstawie tych już istniejących. Klasa dziedzicząca będzie posiadać te same pola i metody co klasa, po której następuje dziedziczenie.
- ***Polimorfizm*** – wołanie różnych implementacji tej samej metody, w zależności od obiektu, który ją woła.

Python ćwiczenia – programowanie obiektowe

Jeszcze jedna rzecz o przeładowaniu operatorów…

Python wie co dodajemy do siebie:

```
In [50]: "hello" + " " + "world"
Out[50]: 'hello world'
```

```
In [51]: 2 + 3 + 4
Out[51]: 9
```

W jakiś sposób wie, jaką operację wykonać w przypadku dodawania różnych obiektów do siebie. Jak to robi?

Python ćwiczenia – programowanie obiektowe

Każda klasa może mieć swoją definicję dodawania, odejmowania, mnożenia, porównywania itd.

Osiąga się to poprzez implementację w ciele klasy specjalnych metod zwanych *magic methods*.

Przykłady:
__add__, __sub__, __mul__, __eq__, __len__, __lt__, __gt__, __init__

Python ćwiczenia – programowanie obiektowe

Bez definicji dodawania w klasie, Python zwraca błąd:

```
In [52]: class Skladnik:
    ...:     def __init__(self, skladnik):
    ...:         self.skladnik = skladnik
    ...:

In [53]: skl1 = Skladnik(10)

In [54]: skl2 = Skladnik(20)

In [55]: skl1 + skl2
---------------------------------------------------------------
TypeError                 Traceback (most recent call last)
<ipython-input-55-4549e37d44f8> in <module>
----> 1 skl1 + skl2

TypeError: unsupported operand type(s) for +: 'Skladnik' and 'Skladnik'
```

Python ćwiczenia – programowanie obiektowe

Z definicją (przeładowaniem) operatora dodawania (__add__), obiekty mogą być dodane zgodnie z naszą implementacją:

```
In [56]: class Skladnik:
    ...:     def __init__(self, skladnik):
    ...:         self.skladnik = skladnik
    ...:     def __add__(self, obj):
    ...:         return self.skladnik + obj.skladnik
    ...:
    ...:

In [57]: skl1 = Skladnik(10)

In [58]: skl2 = Skladnik(20)

In [59]: skl1 + skl2
Out[59]: 30
```

Zróbmy sobie proste ćwiczenie myślowe…

Klasy i metody abstrakcyjne - ABC

- Czasami klasy bazowe, po których dziedziczymy, informują jedynie jakie metody muszą być zaimplementowane w klasach pochodnych ale nie dostarczają konkretnej implementacji.
- Takie klasy nazywamy abstrakcyjnymi. Dokładnie każda klasa, która posiada co najmniej jedną abstrakcyjną metodę (taką, która posiada jedynie sygnaturę ale nie dostarcza jej implementacji) jest nazywana abstrakcyjną.
- Nie można stworzyć bezpośrednio obiektu klasy, która jest abstrakcyjna, najpierw trzeba dostarczyć konkretną klasę, dziedziczącą po abstrakcyjnej, która implementuje wszystkie abstrakcyjne metody.
- Np. możemy stworzyć abstrakcyjną klasę reprezentującą instrument muzyczny. Wiemy, że każdy instrument potrafi jakoś grać i to jest nasza abstrakcyjna metoda.
- Każdy konkretny instrument będzie dostarczał implementacji abstrakcyjnej metody **play**.

```python
import abc


class MusicalInstrument(abc.ABC):

    @abc.abstractmethod
    def play(self):
        pass
```

Autor: Michał Nowotka

Klasy i metody abstrakcyjne - ABC

- Do stworzenia abstrakcyjnej klasy jest nam potrzebny standardowy moduł **abc** (**A**bstract **B**ase **C**lasses).
- Klasa abstrakcyjna powinna dziedziczyć po **abc.ABC** aby zaznaczyć fakt, że jest abstrakcyjna.
- Ponadto powinna posiadać co najmniej jedną metodę udekorowaną dekoratorem **@abc.abstractclass**. W naszym przypadku jest to metoda **play**. Jak widać, nie dostarczamy żadnej implementacji tej metody, od razu piszemy **pass**.
- Chcemy stworzyć konkretną klasę **Guitar** dlatego w deklaracji, klasy zaznaczamy, że dziedziczymy z **MusicalInstrument**. Jednak nasza definicja nie jest poprawna, co sygnalizuje nam PyCharm podkreślając nazwę naszej klasy.
- Problem polega na tym, że nie dostarczyliśmy w klasie **Guitar** konkretnej implementacji metody **play**.

```
1   import abc
2
3
4   class MusicalInstrument(abc.ABC):
5
6       @abc.abstractmethod
7       def play(self):
8           pass
```

```
11  class Guitar(MusicalInstrument):
12      pass
```

Autor: Michał Nowotka

Klasy i metody abstrakcyjne – definicja metody abstrakcyjnej

```python
from abc import ABC, abstractmethod

class MusicalInstrument(ABC):
    @abstractmethod
    def play(self):
        pass
```

lub

```python
from abc import ABC, abstractmethod

class MusicalInstrument(ABC):
    @abstractmethod
    def play(self):
        ...
```

Klasy i metody abstrakcyjne - ABC

- Teraz nasza konkretna klasa jest zdefiniowana poprawnie - gitara mówi nam konkretnie jakie dźwięki z siebie wydaje.
- Ponadto PyCharm na marginesie prezentuje nam przyciski, dzięki którym możemy się wygodnie przełączać pomiędzy abstrakcyjną deklaracją metody a jej konkretnymi implementacjami w klasach pochodnych - nawet jeśli klasy są w oddzielnych plikach.

```
In [1]: paste
import abc


class MusicalInstrument(abc.ABC):

    @abc.abstractmethod
    def play(self):
        pass


class Guitar(MusicalInstrument):
    def play(self):
        return "Brzdęk, brzdęk"
## -- End pasted text --

In [2]: some_instrument = MusicalInstrument()
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-2-a5b6411af71c> in <module>
----> 1 some_instrument = MusicalInstrument()

TypeError: Can't instantiate abstract class MusicalInstrument with abstract methods play

In [3]: my_gibson = Guitar()

In [4]: my_gibson.play()
Out[4]: 'Brzdęk, brzdęk'
```

```
1   import abc
2
3
4   class MusicalInstrument(abc.ABC):
5
6       @abc.abstractmethod
7       def play(self):
8           pass
9
10
11  class Guitar(MusicalInstrument):
12      def play(self):
13          return "Brzdęk, brzdęk"
```

- Na przykładzie jasno widać, że nie da się stworzyć obiektu klasy abstrakcyjnej.

Klasy i metody abstrakcyjne - ABC

- Możemy stworzyć więcej klas instrumentów i zestawić je w orkiestrę!
- Możemy ponadto stworzyć funkcję, która dyryguje instrumentami.
- Najlepsze jest to, że funkcja wcale nie musi wiedzieć, jakimi konkretnie instrumentami dyryguje. Wszystko co jest wystarczy to wiedza, że każdy instrument, aby mógł nazywać się instrumentem, musi dostarczać metodę **play**, więc można ją wykonać i wtedy instrument zagra w sposób specyficzny dla swojego rodzaju.
- Ta możliwość jednolitego traktowania obiektów różnych klas o wspólnej bazie jest nazywana **polimorfizmem**, który oprócz dziedziczenia i abstrakcji jest kolejną ważną koncepcją programowania obiektowego.

```
12  class Guitar(MusicalInstrument):
13      def play(self):
14          return "Brzdęk, brzdęk"
15
16
17  class Flute(MusicalInstrument):
18      def play(self):
19          return "Fiu, fiu!"
20
21
22  class Violin(MusicalInstrument):
23      def play(self):
24          return "Skrzyp, skrzyp!!"
25
26
27  def conductor(instruments:
28                typing.Sequence[MusicalInstrument]
29                ) -> None:
30      for instrument in instruments:
31          print(instrument.play())
```

Autor: Michał Nowotka

Klasy i metody abstrakcyjne - ABC

- Możemy stworzyć więcej klas instrumentów i zestawić je w orkiestrę!
- Możemy ponadto stworzyć funkcję, która dyryguje instrumentami.
- Najlepsze jest to, że funkcja wcale nie musi wiedzieć, jakimi konkretnie instrumentami dyryguje. Wszystko co jest wystarczy to wiedza, że każdy instrument, aby mógł nazywać się instrumentem, musi dostarczać metodę **play**, więc można ją wykonać i wtedy instrument zagra w sposób specyficzny dla swojego rodzaju.
- Ta możliwość jednolitego traktowania obiektów różnych klas o wspólnej bazie jest nazywana polimorfizmem, który oprócz dziedziczenia i abstrakcji jest kolejną ważną koncepcją programowania obiektowego.

```
In [2]: orchestra = [Guitar(), Violin(), Flute()]

In [3]: conductor(orchestra)
Brzdęk, brzdęk
Skrzyp, skrzyp!!
Fiu, fiu!
```

Python ćwiczenia

Napisz program symulujący zoo.
Kod powinien się składać z klasy Zoo (na podstawie orkiestry z przykładu), posiadającej listę zwierząt typu Animal (napisz do trzech klas reprezentujących trzy dowolnie wybrane zwierzęta powinny one dziedziczyć po klasie Animal). Zaimplementuj w klasach metody odpowiedzialne za poruszanie się, dawanie głosu i odżywianie zwierząt.

Kod reprezentujący zachowywanie się zwierząt (przykładowy):

```
>> burek = Dog()
>> burek.make_noise()
”Hau hau!”
>> nemo = Fish()
>> nemo.make_noise()
”Bul bul bul”
```

Wykorzystaj inwencję twórczą ☺