podstawy

October 19, 2021

1 Podstawy języka Python

1.1 17 października 2021 r.

[ ]:
```
sides = [5, 6, 7]

# calculate the semi-perimeter
s = sum(sides) / 2

# calculate the area
area = s
for side in sides:
    area = area * (s - side)
area = area ** 0.5

print(f'The area of the triangle is {area}')
```

1.1.1 Zmienne

[2]:
```
user = "t.dwojak"
mail_domain = "amu.edu.pl"
```

[3]:
```
user1 = "Alicja"
user2 = "Bartosz"
user3 = "Cecylia"
```

[ ]:
```
user = "t.dwojak"
mail_domain = "amu.edu.pl"

email = user + '@' + mail_domain
```

1.1.2 Zmienne

- Nazwy zmiennych muszą być unikatowe.
- Wielkość liter w nazwie zmiennych ma znaczenie.
- Brak konieczności określenia typu.

1.1.3 Funkcja `print`

[6]:
```
print('Hello Python!')
```

```
Hello Python!
```

[8]:
```
print('Hello')
print('Python')
print('Hello', 'Python', '!')
```

```
Hello
Python
Hello Python !
```

[10]:
```
user = 'tomasz'

print(user)
print('Użytkownik:', user)
```

```
tomasz
Użytkownik: tomasz
```

1.1.4 Typy liczbowe

- liczby całkowite: `int`
- liczby rzeczywiste (zmiennoprzecinkowe): `float`

[11]:
```
year = 2021
pi = 3.14159
```

[13]:
```
day = 17
month = 10
year = 2021

print('Dziś jest', day, '/', month, '/', year)
```

```
Dziś jest 17 / 10 / 2021
```

Operacje arytmetyczne na liczbach: * dodawanie `+`, np. `2 + 3` * odejmowanie `-`, np. `10-9` * mnożenie `*`, np. `2.0 * 3.0` * dzielenie `/`, np. `3 / 4` (== 0.75) * dzielenie całkowite `//`, np. `3 / 4` (0) * reszta z dzielenia `%`, np. `6 % 4` (2) * potęgowanie `**`, np. `10 ** 3` (1000) * nawiasy `(...)`

[16]:
```
hour = 10
minutes = 14
seconds = ((60 * 60 * hour) +  60 * minutes)
print("Dziś upłynęło", seconds, "sekund.")
```

```
Dziś upłynęło 36840 sekund.
```

[19]:
```
print( 1 + (20 // 3) + (4 * -5) % 6)
```

```
11
```

1.1.5 Operacje na zmiennej

[20]:
```
x = 5

x = x + 5

x += 5
```

1.1.6 Operatory

Na przykład: * `+=` * `-=` * `/=` * `*=`

1.1.7 Konwersja typów

[23]:
```
pi_int = int(-3.14)
print(pi_int)

trzy = float(3)
print(trzy)
print(3)
```

```
-3
3.0
3
```

[25]:
```
x = '3.14'
print(float(x) * 2)

print(int('42') / 6)
```

```
6.28
7.0
```

1.1.8 Wartości logiczne

W Pythonie są dwie wartości logiczne: * prawda (`True`), * fałsz (`False`).

[29]:
```
prawda = False
print(prawda)
```

```
False
```

- Wszystkie liczby różne od 0 mają wartość logiczną `True`.
- Funkcja `bool` pozwala na konwersję do zmiennej logicznej.
- Do negacji służy słowo kluczowe `not`.

[31]:
```
suma = 1 + 2

print('Wartość logiczna sumy:', bool(suma))

print('Wartość logiczna zera to: ', bool(0), bool(0.0))

print(not False)
```

```
Wartość logiczna sumy: True
Wartość logiczna zera to:  False False
True
```

1.1.9 Operatory porównania

- równość: `==`
- różne: `!=`
- większy: `>`
- większy lub równy: `>=`
- mniejszy: `<`
- mniejszy lub równy `<=`

[38]:
```
user = 'bob'

print(user == 'bob')

print(3 < 7)

print(0 != 0.0)
```

```
True
True
False
```

Do łączenia warunków logicznych służą dwa słowa kluczowe: * `and`: koniunkcja, * `or`: alternatywa.

[39]:
```
print(True and True)
print(True and False)
print(True or False)
print(False or False)
```

```
True
False
True
False
```

W Pythonie istnieje jeszcze jeden typ wartości `None`, który oznacza brak wartości lub wartość pustą.

[40]:
```
user = None
```

```
print('Wartość zmiennej:', user)
print('Wartość logiczna zmiennej:', bool(user))
print(user == False)
```

```
Wartość zmiennej: None
Wartość logiczna zmiennej: False
False
```

1.1.10 Czas na pierwsze zadanie (1a i 1b).

1.2 Komentarze

- Komentarze nie są interpretowane.
- Komentarze w Pythonie zaczynają się od znaku '#'
- Istnieją komentarze wielolinijkowe tagowane potrójnym ", czyli""" """

[41]:
```
print("Bardzo ważna wiadomość") # A to jest komentarz
"""
Komentarz
wielo-
linijkowy
"""
# print("Nie chcę być wydrukowanym")
print("A teraz chcę")
```

```
Bardzo ważna wiadomość
A teraz chcę
```

2 Ciągi znaków (łańcuchy znakowe lub stringi)

- Możemy zdefiniować je na 3 sposoby: `''`, `""` lub `str`.
- Python preferuje pojedynczy cudzysłów.
- Domyślne kodowanie to UTF-8.

[45]:
```
user = 'Tomasz'
user = "Tomasz"
user = str('Tomasz')
```

[49]:
```
sent = "It's fine."

sent = 'It\'s fine.'

print(sent)
```

```
It's fine.
```

[51]:
```
var = str(2021)
```

```
print('rok:', var)
print(var == 2021)
print(var == '2021')
```

```
rok: 2021
False
True
```

[52]:
```
pusty = ''
```

2.0.1 Operacje na stringach

- łączenie: `+`
- powtórzenie: `*`

[53]:
```
username = 't.dwojak'
domain = 'amu.edu.pl'
email = username + '@' + domain
print(email)
```

```
t.dwojak@amu.edu.pl
```

[56]:
```
big_no = 'N' + 'O' * 8 + '!'
print(big_no)
```

```
NOOOOOOOO!
```

[47]:
```
date = str(2021) + '/' + str(10) + '/' + str(17)
print('Dziś jest', date)
```

```
Dziś jest 2021/10/17
```

2.0.2 Operacje na stringach, cd.

- długość łańcucha znakowego: `len`: np. `len('Ala') == 3`,
- zamiana na małe litery `lower` lub na wielkie: `upper`,
- zamiana liter: `replace`,
- usuwanie białych znaków: `strip`,
- sprawdzenie czy string rozpoczyna sie danych prefiksem: `startswith`.

[64]:
```
user = 'mickiewicz'
print('Słowo', user, 'ma', len(user), 'liter.')

print('Python'.lower())

print(user.replace('T', 'R'))

print('      2021     '.strip())
```

```
print(user.startswith('mic'))
```

```
Słowo mickiewicz ma 10 liter.
python
mickiewicz
2021
True
```

[67]:
```
user = 'tomasz'
print('Słowo', user, 'ma', len(user), 'liter.')

print(f'Słowo {user} ma {len(user)} liter.')


print(len(str(123)))
```

```
Słowo tomasz ma 6 liter.
Słowo tomasz ma 6 liter.
3
```

2.0.3 Czas na zadanie (1c).

2.1 Listy (`list`)

- Typ danych, który pozwala przechowywać wiele wartości.
- Dostęp do poszczególnych elementów jest przez indeks elementu.
- Indeksowanie zaczyna się od 0.
- Funkcja `list` zamienia obiekt na listę.

[1]:
```
x = [] # albo równoważnie
y = list()
oceny = [5, 4, 3, 5, 5]
misc = [3.14, "pi", ["pi"], 3]

list_0_9 = list(range(10))
```

[68]:
```
numbers = [6, 7, 9, 11]
print('Liczba elementów:', len(numbers))
```

```
Liczba elementów: 4
```

[73]:
```
numbers = [6, 7, 9, 11]

print(numbers[1])
```

```
7
```

2.2 Dodawanie i usuwanie elementów z listy

Istnieją dwie metody: * `append(x)`: dodaje x na koniec listy * `extend(x)`: rozszerza listę o każdy element z x

[74]:
```
engines = []

engines.append('duck-duck-go')
engines.append("yahoo")
print(engines)
```

```
['duck-duck-go', 'yahoo']
```

[77]:
```
engines = ['duck-duck-go', 'yahoo']
searches = ["google", 'bing']
engines.extend(searches)
print(engines)
```

```
['duck-duck-go', 'yahoo', 'google', 'bing']
```

[78]:
```
engines = ['duck-duck-go', 'yahoo']
searches = ["google", 'bing']

print(engines + searches)
print(engines)
```

```
['duck-duck-go', 'yahoo', 'google', 'bing']
['duck-duck-go', 'yahoo']
```

[83]:
```
liczby = [1, 2, 3, 2, 3, 1, 2, 4]
liczby.pop(1) # Domyślnie usuwa ostatni element z listy
print(liczby)
liczby.remove(2)
print(liczby)
```

```
[1, 3, 2, 3, 1, 2, 4]
[1, 3, 3, 1, 2, 4]
```

Inne przydatne metody: * `sort()`: sortuje listę rosnąco * `count(x)`: zlicza wystąpienia x w liście * `index(x)`: zwraca indeks pierwszego wystąpienia x

[85]:
```
liczby = [1,2,3,2,3,1,2,4]
print(liczby.count(1))
print(liczby.index(4))
liczby.sort()
print(liczby)
```

```
2
7
[1, 1, 2, 2, 2, 3, 3, 4]
```

2.3 Indeksowanie

[87]:
```
oceny = [1, 3, 2, 3, 1, 2, 4]
print('pierwszy element:', oceny[0])
print('ostatni element:', oceny[-1])
print('5 pierwszych:', oceny[:5])
print('5 ostatnich', oceny[-5:])
print('od drugiego, do piątego', oceny[1:5])
print('parzyste:', oceny[1:6:2])
print('od tyłu', oceny[::-1])
```

```
pierwszy element: 1
ostatni element: 4
5 pierwszych: [1, 3, 2, 3, 1]
5 ostatnich [2, 3, 1, 2, 4]
od drugiego, do piątego [3, 2, 3, 1]
parzyste: [3, 3, 2]
od tyłu [4, 2, 1, 3, 2, 3, 1]
```

2.3.1 Funkcje wbudowane

- `len` - zwraca liczbę elementów listy.
- `min` - zwraca wartość najmniejszgo elementu.
- `max` - zwraca wartość największego elementu.
- `sum` - zwraca sumę elementów.
- `all` - zwraca `True`, gdy wszystkie elementy mają wartość `True`.
- `any` - Zwraca `True`, gdy przynajmniej jeden element ma wartość `True`.

[88]:
```
numbers = [4, 8, 12, 18, 0,  32]

print('Liczba elementów:', len(numbers))
print('Najmniejszy element:', min(numbers))
print('Największy element:', max(numbers))
print('Suma elementów:', sum(numbers))

print(all(numbers))
print(any(numbers))
```

```
Liczba elementów: 6
Najmniejszy element: 0
Największy element: 32
Suma elementów: 74
False
True
```

2.3.2 Krotki (`tuple`)

Podobnym typem do listy jest krotka (`tuple`): * definiuje się ją `()` lub `tuple()`, * nie można zmieniać krotki: nie można dodawać ani usuwać elementów, * nie można również zmieniać elemen-

tów, Indeksowanie identyczne jak w listach.

[91]:
```
numbers = (4, 5, 7)

numbers[2]
```

[91]: 7

[92]:
```
users = ([0], [1, 2], [3, 4, 5])

users[0].append(1)

print(users)
```

```
([0, 1], [1, 2], [3, 4, 5])
```

2.3.3 Czas na zadanie (2a, 2b, 2c).

2.3.4 Słowniki (`dict`)

Pewnego rodzaju uogólnieniem listy jest słownik (`dict`), który przechowuje dane jako `klucz`: `wartość`. * Słowniki pozwala na dodawanie, usuwanie i zmianę elementów. * Definiujemy jako `{}` lub `dict()`. * Klucze słownika muszą być niezmienialne (haszowalne).

[93]:
```
store = {}
store = dict()
s_oceny = {
    "Justyna": [5,5,5],
    "Bartek": [3,4,5],
    "Ola": [3,3,3]}
s_oceny = dict([("Justyna", [5,5,5]), ("Bartek", [3,4,5]), ("Ola", [3,3,3])])
```

[26]:
```
s_oceny = {"Justyna" : [5,5,5], "Bartek" : [3,4,5], "Ola": [3,3,3]}

print(s_oceny)
```

```
{'Justyna': [5, 5, 5], 'Bartek': [3, 4, 5], 'Ola': [3, 3, 3], 'Jan': [3, 4, 5]}
```

[94]:
```
s_oceny = {"Justyna" : [5,5,5], "Bartek" : [3,4,5], "Ola": [3,3,3]}
user = 'Bartek'
print(s_oceny[user])
```

```
[3, 4, 5]
```

[95]:
```
s_oceny = {"Justyna" : [5,5,5], "Bartek" : [3,4,5], "Ola": [3,3,3]}

s_oceny['Jan'] = [4, 4, 5]

print(s_oceny)
```

```
{'Justyna': [5, 5, 5], 'Bartek': [3, 4, 5], 'Ola': [3, 3, 3], 'Jan': [4, 4, 5]}
```

[98]:
```
s_oceny = {"Justyna" : [5,5,5], "Bartek" : [3,4,5], "Ola": [3,3,3]}

s_oceny['Ola'].extend([1,4])

print(s_oceny)
```

```
{'Justyna': [5, 5, 5], 'Bartek': [3, 4, 5], 'Ola': [3, 3, 3, 1, 4]}
```

[27]:
```
s_oceny = {"Justyna" : [5,5,5], "Bartek" : [3,4,5], "Ola": [3,3,3]}

del s_oceny["Justyna"]

print(s_oceny)
```

```
{'Bartek': [3, 4, 5], 'Ola': [3, 3, 3]}
```

2.3.5 Operacje

- Funkcja `len` zwraca liczbę elementów w słowniku.
- Domyślnie operacje funkcji wykonywane są na kluczach.
- metoda `keys()` zwraca klucze słownika, `values()` – wartości, a `items()` – pary (klucz, wartość).

[31]:
```
binary_ones = {1: 1, 3: 2, 5: 2, 11: 3}
max(binary_ones)

print(binary_ones.keys())
print(binary_ones.values())
print(binary_ones.items())
```

```
dict_keys([1, 3, 5, 11])
dict_values([1, 2, 2, 3])
dict_items([(1, 1), (3, 2), (5, 2), (11, 3)])
```

[32]:
```
s_oceny = {"Justyna" : [5,5,5], "Bartek" : [3,4,5], "Ola": [3,4,3]}

print(s_oceny["Ola"][1])
```

```
4
```

[33]:
```
users = {'ali99': {'name': 'Alice', 'age': 28}, 'bob90': {'name': 'Bob', 'age':␣
↪19}}

print(users['ali99']['age'])
```

```
28
```

2.3.6 Czas na zadanie (3).

2.4 Instrukcja warunkowa (`if ... elif ... else`)

- Pozwala na wykonanie (wciętego) fragmentu kodu w zależności od czy podany warunek jest spełniony:

```
if <warunek>:
  instrukcja 1
  ...
```

- `elif` pozwala na sprawdzenie kolejnego warunku.
- `else` zostanie wykonany, gdy żaden warunek nie został spełniony.
- Elementy `elif` i `else` nie są obowiązkowe.

[36]:
```
score_theory = 40
score_practical = 45

if score_theory + score_practical > 100:
    print("Zdobyłeś wystarczającą liczbę punktów.")
    print('------')
```

Jedną ważną rzeczą jest to, że kod, który następuje po instrukcji `if` jest wcięty. Dzięki temu interpreter Pythona jest w stanie określić, które instrucje wchodzą w skład bloku warunkowego, a które nie.

Blok `else` jest opcjonalny i jest wykonywany, gdy warunek zawarty w instrukcji `if` nie jest spełniony.

Na poniższym przykładnie mamy własnie taki przypadek. Warunek w `if`ie: `score_theory + score_practical > 100` nie jest spełniony, stąd wykonają się instrukcje z bloku `else`.

[1]:
```
score_theory = 40
score_practical = 45

if score_theory + score_practical > 100:
    print("Zdobyłeś wystarczającą liczbę punktów.")
    print('------')
else:
    print("Nie zdobyłeś wystarczającej liczby punktów.")
```

```
Nie zdobyłeś wystarczającej liczby punktów.
```

Instrukcja warunkowa `if` pozwala na sprawdzenie wielu warunków – służy do tego instrukcja `elif` (else if). Warunki są sprawdzane po kolei (od góry): jeżeli warunek zawarty po `if` nie jest spełniony, wtedy sprawdzany jest pierszy warunek z `elif`. Gdy żaden warunek nie jest spełniony, wtedy wykona się kod zawarty w bloku `else`.

[40]:
```
score_theory = 40
score_practical = 45

```

```
if score_theory + score_practical > 100:
    print("Zdobyłeś wystarczającą liczbę punktów.")
    print('------')
elif score_theory + score_practical > 80:
    print("Będziesz mieć dodatkowy egzamin.")
else:
    print("Nie zdobyłeś wystarczającej liczby punktów.")
```

```
Będziesz mieć dodatkowy egzamin.
```

Nic nie stoi na przeszkodzie, żeby zagnieżdżać instrukcje warunkowe. W poniższym przykładzie widzimy, że druga instrukcja warunkowa jest zawarta w bloku po pierwszym `if`.

[42]:
```
score_theory = 55
score_practical = 50

if score_theory + score_practical > 100:
    print("Zdobyłeś wystarczającą liczbę punktów.")
    if score_theory > score_practical:
        print('Wolisz teorię od praktyki')
```

```
Zdobyłeś wystarczającą liczbę punktów.
Wolisz teorię od praktyki
```

2.4.1 Sprawdzenie obecności w kolekcji (`in`)

- Słowo kluczowe `in` sprawdza czy dany element znajduje sie w kolekcji (np. lista, słownik).
- Jego negacja to `not in`.

Python zawiera bardzo przydatny warunek `in`, który sprawdza czy dany element jest zawarty w danej kolecji, np. w liście lub w słowniku.

W poniższym przykładzie sprawdzamy, czy liczba `67` jest zawarta w liście.

[ ]:
```
numbers = [67, 101, 303]

if 67 in numbers:
    print('67 jest na liście.')
```

W przypadku słowników (`dict`) sprawdzane jest czy istnieje w słowniku zadany klucz.

[44]:
```
ingredients = {'apple': 4, 'lemon': 1, 'cherry': 14}

if 'apple' in ingredients:
    print('Jabłko jest składnikiem.')
```

```
Jabłko jest składnikiem.
```

Możemy wykorzystać warunek `in` do sprawdzenie, czy dany tekst jest podciągiem w drugim.

[43]:
```
my_programming_lang = "Python"

if "Pyt" in my_programming_lang:
    print('Yes!')
```

```
Yes!
```

Żeby sprawdzić czy dany element **nie występuje** możemy wykorzystać instrukcję `not in`.

[45]:
```
shopping_list = ['apples', 'bread', 'carrots']

if 'cookies' not in shopping_list:
    print('Omiń alejkę ze słodyczami.')
```

```
Omiń alejkę ze słodyczami.
```

2.4.2 Obiekty o wartości logicznej `False`

- `0`
- `0.0`
- `[]`
- `()`
- `{}`
- `None`

Bardzo często można spotkać się, gdy chcemy sprawdzić np. czy data lista jest pusta.

[46]:
```
numbers = []

if numbers:
    print('średnia liczb to:', sum(numbers) / len(numbers))
```

2.4.3 Czas na zadanie (4a, 4b).

2.5 Pętla typu `for`

- W Pythonie pętla *for* działa jak pętla *for each* w innych językach;

```
for zmienna in kolekcja:
    instukcja 1
    instrukcja 2
    ...
```

- Pętla pozwala na zapętlenie kodu, który znajduje w wciętym bloku.
- Konstrukcja pętli jest następująca: po słowie `for` musimy podać nazwę zmiennej, która po kolei będzie przyjmować wartości z kolekcji, np. z listy. Pętla wykona się tyle razy, ile jest elementów w kolekcji.
- Funkcja `range(n)` zwraca kolekcję liczb od `0` do `n-1`.

To na co warto zwrócić uwagę to wcięcie. Kod, który ma zostać wykonany w pętli musi być wcięty.

[48]:
```
ingredients = ['apples', 'cherries', 'pineapple']

for ingredient in ingredients:
    print('element', ingredient)
```

```
element apples
element cherries
element pineapple
```

Funkcja `range(n)` zwraca obiekt, który możemy przekonwertować na listę elementów od 0 do `n-1`.

[3]:
```
print(list(range(5)))
```

```
[0, 1, 2, 3, 4]
```

Bardzo często można spotkać poniższą kombinację funkcji `range` i `len` w kontekście pętli:

[47]:
```
ingredients = ['apples', 'cherries', 'pineapple']

for i in range(len(ingredients)):
    print(ingredients[i])
```

```
apples
cherries
pineapple
```

W przypadku połączenia słownika i pętli, do zmiennej przypisywane są kolejne klucze ze słownika (przykład poniżej):

[49]:
```
shopping_list = {'apple': 4, 'lemon': 1, 'cherry': 14}
for item in shopping_list:
    print(item)
```

```
apple
lemon
cherry
```

Jak wcześniej wspomniałem, metoda `items()` zwraca listę krotek: `(klucz, wartość)` i możemy wykorzystać to w pętli `for`. W tym przypadku musimy podać dwie zmienne:

[51]:
```
shopping_list = {'apples': 4, 'lemon': 1, 'cherries': 14}
for item, number in shopping_list.items():
    print('Buy', number, item)
```

```
Buy 4 apples
Buy 1 lemon
Buy 14 cherries
```

Możemy też iterować po stringu – znak po znaku:

[52]:
```
for char in 'Python':
    print(char)
```

```
P
y
t
h
o
n
```

Ćwiczenie: czy rozumiesz kod, który był na początku zajęć?

[53]:
```
sides = [5, 6, 7]

# calculate the semi-perimeter
s = sum(sides) / 2

# calculate the area
area = s
for side in sides:
    area = area * (s - side)
area = area ** 0.5

print(f'The area of the triangle is {area}')
```

```
The area of the triangle is 14.696938456699069
```

Zagnieżdżając dwie pętle musimy pamiętać o dodatkowym wcięciu (tak jak w przykładzie poniżej).

[64]:
```
for i in range(3):
    for j in range(i+1):
        print((i + 1) * (j + 1))
```

```
1
2
4
3
6
9
```

2.5.1 Czas na zadanie (5a, 5b, 5c i 5d).

2.5.2 Funkcje

- Pozwalają na uniknięcie pisania tego samego kodu za każdym razem.
- Pozwalają na wielokrotne wykorzystanie tego samego fragmentu kodu.
- Zwiększają czytelność kodu.

Definicja funkcji wygląda następująco: * najpierw piszemy słowo kluczowe `def`, następnie nazwę funkcji, później w nawiasach okrągłych listę argumentów i kończymy dwukropkiem. * następnie

wcięty blok to będzie kod funkcji.

[5]:
```
def hello():
    print('Hello!')

hello()
```

```
Hello!
```

W powyższym przykładzie zdefiniowaliśmy funkcje `hello`, która nie przyjmuje żadnych argumentów. Wywołanie tej funkcji nie różni się od wywołań innych funkcji.

Słowo kluczowe `return` pozwala na zwrócenie wartości przez funkcję. Poniższa funkcja `get_five` zwraca liczbę pięć.

[6]:
```
def get_five():
    return 5
x = get_five()
print(x)
```

```
5
```

Argumenty funkcji umieszczamy w nawiasach:

```
def nazwa_funkcji(arg_1, arg_2, arg_3):
    instrukcja 1
    instrukcja 2
    return jakaś wartość
```

Przykład: poniższa funkcja `get_bigger` przyjmuje dwa argumenty: `a` i `b`, które rozdzielamy przecinkiem.

[69]:
```
def get_bigger(a, b):
    if a >= b:
        return a
    return b
print(get_bigger(56, 512))
```

```
512
```

[7]:
```
Przykłady:
```

```
  File "/tmp/ipykernel_3661/250649810.py", line 1
    Przykłady:
              ^
SyntaxError: invalid syntax
```

[ ]:
```
def srednia(lista):
    s = 0
```

```
    for item in lista:
        s += item
    return s / len(lista)

print(srednia([7,8,9]))
```

[6]:
```
def count(lista, item):
    l = 0
    for i in lista:
        if i == item:
            l += 1
    return l
```

Podczas wywoływania funkcji możemy dopowiedzieć, która argument jaką przyjmuje wartość.

[9]:
```
count([5,5,5,4], 5)
count(lista=[5,5,5,4], item=5)
count(item=5, lista=[5,5,5,4])
```

[9]: 3

2.5.3 Czas na zadanie (7a i 7b).

2.6 Korzystanie z bibliotek

Python posiada bogatą kolekcję bibliotek wbudowanych, tzn. które są dostarczone wraz z interpreterem.

Żeby móc wykorzystać daną bibliotekę, to musimy ją zaimportować. Możemy to zrobić na dwa sposoby: * `import <nazwa_biblioteki>`. Dostęp do elementów jest poprzez `<nazwa_biblioteki>.nazwa_funkcji`. * `from <nazwa_biblioteki> import <nazwa funkcji>`: pozwala na dołączenie elementów biblioteki, tak jakby były częścią naszego skryptu.

Przykłady:

[9]:
```
import os
print(os.name)

from os import getenv
print('Nazwa uzytkownika: {}'.format(getenv("USER")))
```

```
posix
Nazwa uzytkownika: tomasz
```

1. Importujemy bibliotekę `os`. Dostęp do stałej jest "przez kropkę".
2. Z biblioreki `os` importujemy funkcję `getenv` (zwraca wartości zmiennych systemowych).

Wystarczy, że zaimportujemy raz daną bibliotekę:

[17]:
```
import math
math.cos(math.pi)
```

[17]:
```
-1.0
```

Jeżeli nazwa biblioteki jest za długa, to możemy użyć aliasu: `import <nazwa_biblioteki> as <alias>`:

[70]:
```
import calendar as cal
cal.TextCalendar().prmonth(2021, 10)
```

```
    October 2021
Mo Tu We Th Fr Sa Su
                 1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
```

Ważniejsze biblioteki wbudowane: * `os`, `sys`: obsługa rzeczy dt. systemu i środowiska * `datetime`: wszystko co jest związane z czasem * `collections`: zawiera `Counter` i `defaultdict`

2.6.1 Czas na zadanie (9a i 9b).