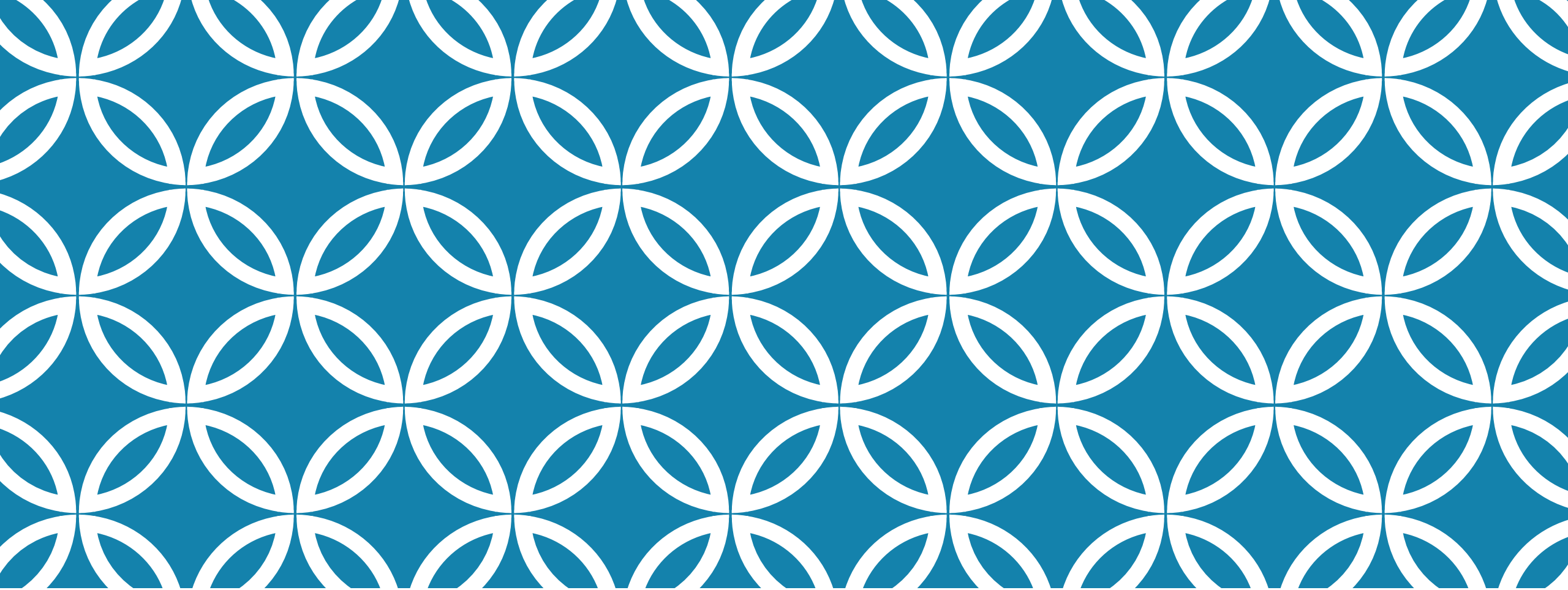


4. THE CORE PYTHON LANGUAGE II

Opracował: Jakub Olszewski

4.1. ERRORS AND EXCEPTIONS

Dwa główne rodzaje błędów:

`SyntaxError`

Błędy w gramatyce wychwycone przez kompilator zanim program zostanie wykonany.

Zawsze zatrzymuje działanie.

`Exception`

Wyjątki spowodowane błędami logicznymi.

Nie przechwycone przez `try...except...` powodują wywołanie informacji o błędzie, zazwyczaj bez zatrzymywania działania programu.

4.1.1. SYNTAX ERRORS

`while` jest słowem kluczowym i nie może być wykorzystane do iteracji

```
>>> for while in range(4):

SyntaxError: invalid syntax
>>>
```

Niezamknięcie nawiasu

```
>>> for i in range(4:

SyntaxError: invalid syntax
>>>
```

Niespodziewane przypisanie wewnątrz nawiasu

```
>>> a = [1,2,3,4
     b = 5

SyntaxError: invalid syntax
>>>
```

4.1.2. EXCEPTIONS

Wyjątki pojawiają się gdy gramatycznie poprawny wykonany kod powoduje `runtime error` – błąd napotkany w trakcie działania programu.

Python zawiera wbudowane wyjątki produkujące komunikaty o błędach, zazwyczaj tworząc `stack traceback` – historię wykonanych poleceń które doprowadziły do błędu – które pomaga w rozwiązywaniu problemów.

```
>>> print(zmienna)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    print(zmienna)
NameError: name 'zmienna' is not defined
>>> float("hello")
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    float("hello")
ValueError: could not convert string to float: 'hello'
>>>
```

```
>>> a, b = 0, 10
>>> b / a
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    b / a
ZeroDivisionError: division by zero
>>> int('7.0')
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    int('7.0')
ValueError: invalid literal for int() with base 10: '7.0'
>>> int(float('7.0'))
7
>>>
```

```python
def funkcja(x):
    def zagniezdzona(y):
        z = 1 / y
        return z
    return zagniezdzona(x)

if __name__ == "__main__":
    x = 0
    funkcja(x)
```

```
Traceback (most recent call last):
  File "C:/Users/kubac/OneDrive/Desktop/przyklad.py", line 9, in <module>
    funkcja(x)
  File "C:/Users/kubac/OneDrive/Desktop/przyklad.py", line 5, in funkcja
    return zagniezdzona(x)
  File "C:/Users/kubac/OneDrive/Desktop/przyklad.py", line 3, in zagniezdzona
    z = 1 / y
ZeroDivisionError: division by zero
>>>
```

Table 4.1 Common Python exceptions

Exception	Cause and description
`FileNotFoundError`	Attempting to open a file or directory that does not exist – this exception is a particular type of `OSError`.
`IndexError`	Indexing a sequence (such as a list or string) with a subscript that is out of range.
`KeyError`	Indexing a dictionary with a key that does not exist in that dictionary (see Section 4.2.2).
`NameError`	Referencing a local or global variable name that has not been defined.
`TypeError`	Attempting to use an object of an inappropriate type as an argument to a built-in operation or function.
`ValueError`	Attempting to use an object of the correct type but with an incompatible value as an argument to a built-in operation or function.
`ZeroDivisionError`	Attempting to divide by zero (either explicitly (using "/" or "//") or as part of a modulo operation "%").
`SystemExit`	Raised by the `sys.exit` function (see Section 4.4.1) – if not handled, this function causes the Python interpreter to exit.

4.1.3. HANDLING AND RAISING EXCEPTIONS

W trakcie pisania programu może okazać się, że musimy obsługiwać się danymi w sposób który może wywołać błąd. Jeżeli nie jest on powodem do zatrzymania programu, ten błąd może zostać przechwycony przez klauzulę `try...except` w celu wykonania określonego działania.

„It is Easier to Ask Forgiveness than to seek Permission” (EAFP)

```
try:
    kod do wykonania
    więcej kodu
    ...
except JakiśError:      #Lub except (JakiśError,InnyError,...):
    kod do wykonania w razie znalezienia *zadeklarowanego*
    błędu wewnątrz try
except InnyError:
    kod do wykonania w razie znalezienia innego zadeklarowanego
    błędu wewnątrz try
else:
    kod do wykonania gdy nie znaleziono błędu wewnątrz try
finally:
    kod do wykonania *zawsze* bo wykonaniu powyższych bloków
```

```
try:
    y = 1 / x
    print('1 /', x, ' = ', y)
except ZeroDivisionError:
    print('1/0 is not defined')
```

4.1.3. HANDLING AND RAISING EXCEPTIONS

Python pozwala również na „ręczne” wywoływanie błędów przy pomocy funkcji `raise`:

```
if n%2:
    raise ValueError('n musi być parzyste!')
```

Podobna funkcja `assert` pozwala na wywołanie AssertionError w przypadku gdy załączony warunek nie jest prawdziwy:

```
>>> assert 2 == 2
>>> assert 1 == 2, '1 nie równa się 2'
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    assert 1 == 2, '1 nie równa się 2'
AssertionError: 1 nie równa się 2
>>>
```

```python
def iloczyn_wektorowy(a, b):
    assert len(a) == len(b) == 3, 'Wektory a i b muszą być 3-wymiarowe'
    return [a[1]*b[2] - a[2]*b[1],
            a[2]*b[0] - a[0]*b[2],
            a[0]*b[1] - a[1]*b[0]]

if __name__ == "__main__":
        a1, b1 = [1,2,-1], [2,0,-1,3]
        a2, b2 = [1,2,-1], [2,0,-1]
```

```
>>> iloczyn_wektorowy(a1,b1)
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    iloczyn_wektorowy(a1,b1)
  File "C:\Users\kubac\OneDrive\Desktop\przyklad.py", line 9, in iloczyn_wektorowy
    assert len(a) == len(b) == 3, 'Wektory a i b muszą być 3-wymiarowe'
AssertionError: Wektory a i b muszą być 3-wymiarowe
>>> iloczyn_wektorowy(a2,b2)
[-2, -1, -4]
>>>
```

4.2. DICTIONARIES AND SETS

`dict` (dictionary, ang. słownik) jest mutowalnym szeregiem par `key-value`, przy czym `key` jest unikalnym obiektem niemutowalnym oraz hashowalnym, a `value` jakimkolwiek innym obiektem.

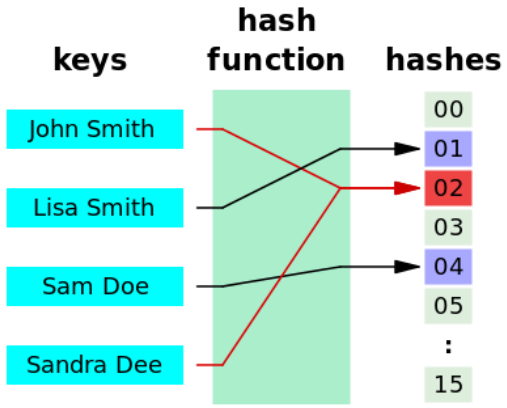
```
>>> d = {'a': 1, 'b': 2, 'c': 3}
>>> d
{'a': 1, 'b': 2, 'c': 3}
>>> d['a']
1
>>>
```

`set` (ang. zestaw, złożenie) jest mutowalnym szeregiem unikalnych hashowalnych elementów.

```
>>> s = set([1,1,4,3,2,2,3,4,1,'hello!'])
>>> s
{1, 2, 3, 4, 'hello!'}
>>>
```

HASH FUNCTION

Funkcja stosowana w informatyce, nadawana obiektom w celu przypisania im krótkich i łatwych do weryfikacji sygnatur pozwalających na szybki i łatwy dostęp.




```
>>> klucz = "John Smith"
>>> klucz.__hash__()
-6321013222125803853
```

W Pythonie hashowalne są niemutowalne obiekty jak `str`, `int`, `float`, `tuple` czy `bool`.

4.2.1. DEFINING AND INDEXING A DICTIONARY

Słownik definiuje się zawierając pary `key: value` wewnątrz nawiasu klamrowego `{}` lub podania tych samych par w postaci listy krotek `(key, value)` do konstruktora `dict`:

```
>>> slownik1 = {'slowo' : 5, 'inne slowo': [1,2,3], 'kolejne': 'itd'}
>>> slownik1
{'slowo': 5, 'inne slowo': [1, 2, 3], 'kolejne': 'itd'}
>>> slownik1['inne slowo']
[1, 2, 3]
>>> slownik2 = dict([('slowo',5),('inne slowo',[1,2,3]),('kolejne','itd')])
>>> slownik1 == slownik2
True
```

Utworzony zostaje iterowalny obiekt:

```
>>> for key in slownik1:
        print(key, '|', slownik1[key])


slowo | 5
inne slowo | [1, 2, 3]
kolejne | itd
```

4.2.2. DICTIONARY METHODS

`get()` – wyciąga podaną wartość dla danego klucza jeżeli istnieje w słowniku, jeżeli nie – zwraca wartość `None` lub wartość domyślną (jeżeli została taka podana).

```
>>> slownik = {'jeden': 1, 'dwa': 2, 'trzy': 3}
>>> slownik['cztery']
Traceback (most recent call last):
  File "<pyshell#22>", line 1, in <module>
    slownik['cztery']
KeyError: 'cztery'
>>> slownik.get('cztery')     # Brak błędu
>>> slownik.get('cztery', -1)# Wyrzuci domyślną wartość -1
-1
>>> print(slownik.get('cztery'))
None
```

4.2.2. DICTIONARY METHODS

`keys()`, `values()` i `items()` – trzy metody zwracające obiekty odnoszące się kolejno do `key`, `value` oraz par `key: value` (jako krotki).

```
>>> slownik = {'jeden': 1, 'dwa': 2, 'trzy': 3}
>>> dictValues = slownik.values()
>>> dictValues
dict_values([1, 2, 3])
>>> dictValues[0]
Traceback (most recent call last):
  File "<pyshell#36>", line 1, in <module>
    dictValues[0]
TypeError: 'dict_values' object is not subscriptable
>>> for value in dictValues:
        print(value)


1
2
3
```

```
>>> dictValues_list = list(dictValues)
>>> dictValues_list[0]
1
>>> for item in slownik.items():
        print(item)


('jeden', 1)
('dwa', 2)
('trzy', 3)
>>>
```

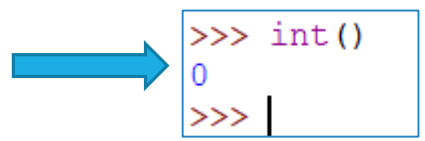
4.2.2. DICTIONARY METHODS - DEFAULTDICT

Przekazując do słownika `key` który nie istnieje w tym słowniku zostanie nam zwrócony wyjątek `KeyError`. Przydatną klasą która obchodzi ten problem jest `defaultdict` z biblioteki `collections`, która przy otrzymaniu nieistniejącego klucza zwraca zadaną domyślną wartość (musi być `callable` – być klasą z metodą `.__call__()`).

```
>>> slownik = {'jeden': 1, 'dwa': 2, 'trzy': 3}
>>> slownik['cztery']
Traceback (most recent call last):
  File "<pyshell#58>", line 1, in <module>
    slownik['cztery']
KeyError: 'cztery'
>>> from collections import defaultdict
>>> defslownik = defaultdict(int)
>>> defslownik['cztery']
0
>>> defslownik
defaultdict(<class 'int'>, {'cztery': 0})
>>>
```

```
>>> int.__call__()
0
>>> str.__call__()
''
```

```python
from collections import defaultdict

text = """Fourscore and seven years ago our fathers brought
        forth, on this continent, a new nation, conceived in
        liberty, and dedicated to the proposition that all
        men are created equal."""

text = text.replace(',','').lower()

word_lengths = {}
for word in text.split():
    try:
        word_lengths[len(word)] += 1
    except KeyError:
        word_lengths[len(word)] = 1
print(word_lengths)

print('### ###')

word_lengths = defaultdict(int)
for word in text.split():
    word_lengths[len(word)] += 1
print(word_lengths)
```




```
Długości słów - zwyczajna metoda:
 {9: 4, 3: 9, 5: 3, 7: 4, 2: 3, 4: 2, 1: 1, 6: 2, 11: 1}
### ###
Długości słów - defaultDict:
 defaultdict(<class 'int'>, {9: 4, 3: 9, 5: 3, 7: 4, 2: 3, 4: 2, 1: 1, 6: 2, 11: 1})
```

4.2.3. SETS

Zbiór definiuje się zawierając listę hashowalnych elementów wewnątrz dwóch nawiasów ({...}) albo przekazując tą listę do konstruktora `set()`:

```
>>> s = ({1,4,1,2,1,5,7,2,1,"hello"})
>>> s
{1, 2, 4, 5, 7, 'hello'}
>>> s = set([1,4,1,2,1,5,7,2,1,"hello"])
>>> s
{1, 2, 4, 5, 7, 'hello'}
>>> s.add('world')
>>> s
{1, 2, 4, 5, 'hello', 7, 'world'}
>>> s.pop()
1
>>> s
{2, 4, 5, 'hello', 7, 'world'}
```

```
>>> s.remove(2)
>>> s
{4, 5, 'hello', 7, 'world'}
>>> s.discard(1000)
>>> s
{4, 5, 'hello', 7, 'world'}
>>> s.remove(1000)
Traceback (most recent call last):
  File "<pyshell#25>", line 1, in <module>
    s.remove(1000)
KeyError: 1000
>>> s.clear()
>>> s
set()
```

```
>>> len(s)        # Moc zbioru
6
>>> 2 in s, 10 in s # Przynależność do zbioru
(True, False)
>>> for item in s:
        print(item)


1
2
4
5
7
hello
>>>
```

```
>>> A = set((1,2,3))
>>> B = set((1,2,3,4))
>>> A <= B
True
>>> A.issubset((1,2,3,4))
True
>>> A | B
{1, 2, 3, 4}
>>> A | B      # Suma
{1, 2, 3, 4}
>>> A & B      # Część wspólna
{1, 2, 3}
>>> B - A      # Różnica
{4}
```

Table 4.2 set methods

	Method	Description
	`isdisjoint(other)`	Is *set* disjoint with *other*?
$A \subseteq B$	`issubset(other)`, `set <= other`	Is *set* a subset of *other*?
$A \subset B$	`set < other`	Is *set* a proper subset of *other*?
	`issuperset(other)`, `set >= other`	Is *set* a superset of *other*?
	`set > other`	Is *set* a proper superset of *other*?
$A \cup B$	`union(other)`, `set | other | ...`	The union of *set* and *other*(s)
$A \cap B$	`intersection(other)`, `set & other & ...`	The intersection of *set* and *other*(s)
$A \setminus B$	`difference(other)`, `set - other - ...`	The difference of *set* and *other*(s)
$A \Delta B$	`symmetric_difference(other)`, `set ^ other ^ ...`	The symmetric difference of *set* and *other*(s)

4.2.3. FROZENSETS

Zbiory są obiektami mutowalnymi oraz są niehashowalne, czyli nie mogą być użyte jako klucz w słowniku albo jako element zbioru:

```
>>> a = set((1,2,3))
>>> b = set(('q',(1,2), a))
Traceback (most recent call last):
  File "<pyshell#84>", line 1, in <module>
    b = set(('q',(1,2), a))
TypeError: unhashable type: 'set'
```

`frozenset()` jest niemutowalnym, hashowalnym odpowiednikiem `set()`:

```
>>> a = frozenset((1,2,3))
>>> b = set(('q',(1,2), a))
>>> b
{frozenset({1, 2, 3}), (1, 2), 'q'}
```

4.3. PYTHONIC IDIOMS: “SYNTACTIC SUGAR”

Wiele języków programowania zapewnia syntax pozwalający na ułatwienie pisania oraz rozumienia kodu. Takiego rodzaju „składniowy cukier” (syntatic sugar) pozbywa się niektórych elementów bez naruszania funkcjonalności języka.

augmented assignment:

```
a += 1
```

```
a = a + 1
```

negative indexing:

```
b[-1]
```

```
b[len(b)-1]
```

4.3.1. COMPARISON AND ASSIGNMENT SHORTCUTS

```
>>> x = y = z = -1
>>> x
-1
>>> y
-1
>>> z
-1
```

```
>>> a = b = c = [1,2,3]
>>> a
[1, 2, 3]
>>> b
[1, 2, 3]
>>> c
[1, 2, 3]
>>> a.append(4)
>>> a
[1, 2, 3, 4]
>>> b
[1, 2, 3, 4]
>>> c
[1, 2, 3, 4]
```

```
>>> e, f, g = 'hello', -4.5, 1
>>> e
'hello'
>>> f
-4.5
>>> g
1
```

```
>>> a = b = 10
>>> a == b == 10
True
>>> a >= b > 1
True
```

```
>>> x = 5
>>> 1 < x < 10
True
```

```
>>> x = 10
>>> import math
>>> y = math.sin(x)/x if x else 1
>>> y
-0.054402111088893698
>>> x = 0
>>> y = math.sin(x)/x if x else 1
>>> y
1
```

```
try:
    y = math.sin(x)/x
except ZeroDivisionError:
    y = 1
```

4.3.2. LIST COMPREHENSION

Jest to konstrukt pozwalający na stworzenie listy na podstawie innego iterowalnego obiektu, np. na podstawie innej listy. Tworzy się go zamieszczając pętlę for wewnątrz nawiasu kwadratowego [ ] :

```
>>> xlist = [1,2,3,4,5,6]
>>> x2list = [x**2 for x in xlist]
>>> x2list
[1, 4, 9, 16, 25, 36]
```

```
>>> x2list_2 = []
>>> for x in xlist:
        x2list_2.append(x**2)


>>> x2list_2 == x2list
True
```

```
>>> x2list = [x**2 for x in xlist if x%2]
>>> x2list
[1, 9, 25]
```

```
>>> [x%2 for x in xlist]
[1, 0, 1, 0, 1, 0]
>>> 1 == True
True
>>> 0 == False
True
```

```
>>> litery = 'abc def'
>>> [l.upper() for l in litery]
['A', 'B', 'C', ' ', 'D', 'E', 'F']
```

```
                  v       v       v
                c, c, c   c, c, c   c, c, c
>>> vlist = [[1,2,3],[4,5,6],[7,8,9]]
>>> [c for v in vlist for c in v]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

4.3.3. LAMBDA FUNCTIONS

`lambda` jest rodzajem prostych anonimowych funkcji które mogą być przypisywane i przechowywane np. wewnątrz list.

```
>>> f = lambda x: x**2 - 3*x + 2
>>> f(4)
6
>>> f = lambda x,y: x**2 + 2*x*y + y**2
>>> f(2,3)
25
```

```
>>> f = lambda x: x+1
>>> type(f)
<class 'function'>
>>> f.__hash__()
186386560525
```

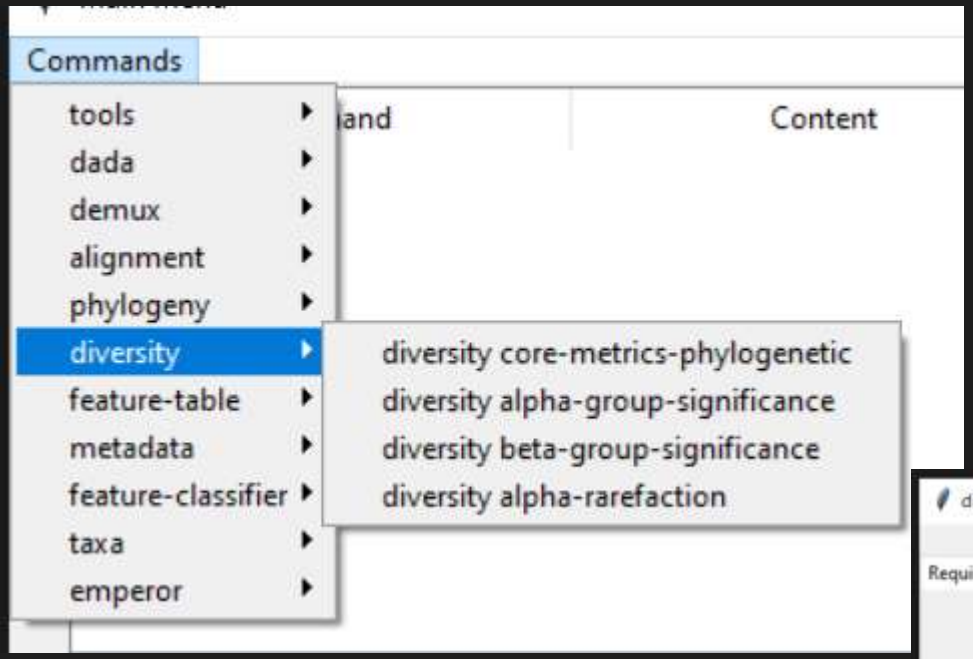
```
>>> flist = [lambda x: 1,
             lambda x: x,
             lambda x: x**2,
             lambda x: x**3]
>>> flist[3](5)      #flist[3] == x**3
125
>>> flist[2](4)      #flist[2] == x**2
16
```

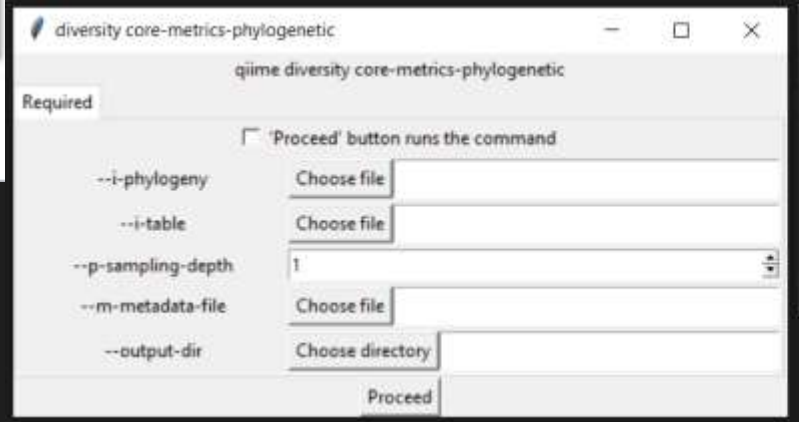
```
>>> sorted('Sorted is a Case Sensitive function'.split())
['Case', 'Sensitive', 'Sorted', 'a', 'function', 'is']
>>> sorted('Sorted is a Case Sensitive function'.split(), key = str.upper)
['a', 'Case', 'function', 'is', 'Sensitive', 'Sorted']
```

```
>>> halogens = [('At', 85),('Br', 35), ('Cl', 17), ('F', 9), ('I', 53)]
>>> sorted(halogens, key = lambda e: e[1])
[('F', 9), ('Cl', 17), ('Br', 35), ('I', 53), ('At', 85)]
```

```python
# Create popup window with command
def popupWig(master,widget,key):
    """Uses winBuild to create a pop-up window with instructions from widget and key.

    :param widget: a nested list of widgets used in the pop-up window for that command
    :type widget: tuple

    :param key: name of the command
    :type key: str
    """
    newWindow = Toplevel(master)
    newWindow.title("{}".format(key))
    global pipelineTreeContent
    pipelineTreeContent = winBuild(newWindow,widget,key)
```

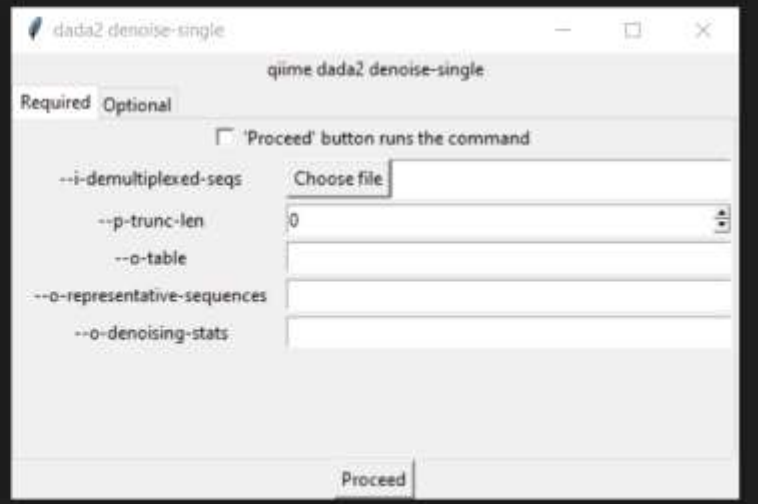
```python
# Menu Bar
commands = Menu(menuBar,tearoff=0)
for package in commandWig.keys():
    subMenu = Menu(commands,tearoff=0)
    for key in commandWig[package]:
        subMenu.add_command(label="{}".format(key),command=lambda packKey=(package,key): popupWig(subMenu,commandWig[packKey[0]][packKey[1]],packKey[1]))
    commands.add_cascade(label="{}".format(package),menu=subMenu,underline=0)
menuBar.add_cascade(label = "Commands",menu = commands)

root.config(menu=menuBar)
```


Commands
tools
dada
demux
alignment
phylogeny
diversity
feature-table
metadata
feature-classifier
taxa
emperor
and
Content
diversity core-metrics-phylogenetic
diversity alpha-group-significance
diversity beta-group-significance
diversity alpha-rarefaction


diversity core-metrics-phylogenetic
qiime diversity core-metrics-phylogenetic
Required
'Proceed' button runs the command
--i-phylogeny
Choose file
--i-table
Choose file
--p-sampling-depth
1
--m-metadata-file
Choose file
--output-dir
Choose directory
Proceed


tools import
qiime tools import
Required
Optional
'Proceed' button runs the command
--type
--input-path
Choose file
Choose dir
--output-path
Proceed


diversity alpha-group-significance
qiime diversity alpha-group-significance
Required
'Proceed' button runs the command
--i-alpha-diversity
Choose file
--m-metadata-file
Choose file
--o-visualization
Proceed


dada2 denoise-single
qiime dada2 denoise-single
Required
Optional
'Proceed' button runs the command
--i-demultiplexed-seqs
Choose file
--p-trunc-len
0
--o-table
--o-representative-sequences
--o-denoising-stats
Proceed

4.3.4. THE „WITH” STATEMENT

`with` otwiera blok kodu który będzie egzekwowany w zadanym przez użytkownika kontekście.

```
f = open('filename.txt', 'w')
f.write('hello')
f.close()
```

```
f = open('filename.txt', 'w')
try:
    f.write('hello')
finally:
    f.close()
```

```
with open('filename.txt', 'w') as f:
    f.write('hello')
```

```
# a simple file writer object

class MessageWriter(object):
    def __init__(self, file_name):
        self.file_name = file_name

    def __enter__(self):
        self.file = open(self.file_name, 'w')
        return self.file

    def __exit__(self):
        self.file.close()

# using with statement with MessageWriter

with MessageWriter('my_file.txt') as xfile:
    xfile.write('hello world')
```

4.3.5. GENERATORS

Generatory są funkcjami które zachowują się jak iterowalny obiekt, tzn. może zostać użyty w pętli for i w każdej iteracji będzie zwracać generowane obiekty. Jest to rozwiązanie bardziej wydajne i oszczędza więcej miejsca niż tradycyjna metoda polegająca na przechowywaniu wartości wewnątrz zmiennych.

```
# Generator liczący do zadanej liczby n
def count(n):
    i = 0
    while i < n:
        i += 1
        yield i


>>> for j in count(5):
    print(j)


1
2
3
4
5
```

```
# Generator Ciągu Fibbonacciego
def fib(limit):
    # Dwie pierwsze liczby
    a, b = 0, 1

    while a < limit:
        yield a
        a, b = b, a + b
```

```
>>> fibo = fib(5)
>>> fibo.__next__()
0
>>> fibo.__next__()
1
>>> fibo.__next__()
1
>>> fibo.__next__()
2
>>> fibo.__next__()
3
>>> fibo.__next__()
Traceback (most recent call last):
  File "<pyshell#98>", line 1, in <module>
    fibo.__next__()
StopIteration
```

4.3.5. GENERATORS – GENERATOR COMPREHENSION

Generatory również można tworzyć używając podobnej składni jak *list comprehention*, używając okrągłych nawiasów ( ) zamiast kwadratowych [ ] :

```
>>> kwadraty = (x**2 for x in range(5))
>>> for kwadrat in kwadraty:
        print(kwadrat)


0
1
4
9
16
```

Należy pamiętać, że po jednorazowym przejściu generator zostaje „wyczerpany”:

```
>>> for kwadrat in kwadraty:
        print(kwadrat)


>>>
```

4.3.6. MAP

Funkcja map zwraca iterator który nakłada zadaną funkcję na każdy obiekt w podanej sekwencji:

```
>>> listy = [[1,2,3],[10,20,30],[100,200,300]]
>>> list(map(sum, listy))
[6, 60, 600]
>>> sumy = map(sum,listy)
>>> type(sumy)
<class 'map'>
>>> for suma in sumy:
        print(suma)


6
60
600
```

4.3.7. ASSIGNMENT EXPRESSIONS: THE WALRUS OPERATOR

Załóżmy, że chcemy stworzyć program który potrzebuje string o długości `<= 10`. Warunek sprawdzający możemy napisać w taki sposób:

```
s = 'String z dużą ilością znaków'
if len(s) > 10:
    print(f's ma {len(s)} znaków. Maksimum to 10.')
```

Aby nie sprawdzać dwa razy długości stringu możemy skorzystać ze zmiennej pomocniczej:

```
s = 'String z dużą ilością znaków'
slen = len(s)
if slen > 10:
    print(f's ma {slen} znaków. Maksimum to 10.')
```

Lub krócej, używając *assignment expression* (`slen := len(s)`):

```
s = 'String z dużą ilością znaków'
if (slen := len(s)) > 10:
    print(f's ma {slen} znaków. Maksimum to 10.')
```

4.4. OPERATING SYSTEM SERVICES

Python pozwala na interakcję z funkcjami dostępnymi w większości systemów operacyjnych takie jak np. pliki czy zegar. Takie operacje zazwyczaj są opierane na systemach Unix oraz interfejsach C, ale są dostępne również w (prawie) wszystkich innych systemach.

UNIX®

4.4.1. THE SYS MODULE

`sys.argv` – jest to lista stringów przesłana do Python'a kiedy program jest wykonywany. Pierwszym elementem `sys.argv[0]` jest nazwą programu, a każdy kolejny element to argumenty przekazane wraz z uruchomieniem programu.

```
# kwadrat.py
import sys

n = int(sys.argv[1])
print(n, 'do kwadratu równa się', n**2)
```

```
C:\Users\kubac\przyklad>python kwadrat.py 10
10 do kwadratu równa się 100

C:\Users\kubac\przyklad>python kwadrat.py 25
25 do kwadratu równa się 625

C:\Users\kubac\przyklad>python kwadrat.py 36
36 do kwadratu równa się 1296
```

4.4.1. THE SYS MODULE

`sys.exit` – sprawia, że program ulega zamknięciu w sposób „czysty” tzn. wszystkie elementy z klauzuli `finally` zostają egzekwowane, a wszystkie otwarte pliki zostają zamknięte. Opcjonalnie, `sys.exit` przyjmuje również jakikolwiek argument który jest przekazywany systemowi.

```
# kwadrat.py
import sys

try:
    n = int(sys.argv[1])
except (IndexError, ValueError):
    sys.exit('Proszę podać interger, <n>, w lini komend.\nSposób korzystania: '
             '\n\tpython {:s} <n>'.format(sys.argv[0]))
print(n, 'do kwadratu równa się', n**2)
```

```
C:\Users\kubac\przyklad>python kwadrat.py
Proszę podać interger, <n>, w lini komend.
Sposób korzystania:
        python kwadrat.py <n>

C:\Users\kubac\przyklad>
```

4.2.2. THE OS MODULE

`os.getenv(key)` – pozwala na dostęp do zmiennych środowiskowych w systemie operacyjnym.

```
>>> import os
>>> os.getenv("APPDATA")
'C:\\Users\\kubac\\AppData\\Roaming'
>>> os.getenv("HOMEDRIVE")
'C:'
>>> os.getenv("PROGRAMDATA")
'C:\\ProgramData'
>>> os.getenv("SYSTEMROOT")
'C:\\Windows'
>>> os.getenv("OneDrive")
'C:\\Users\\kubac\\OneDrive'
>>> os.getenv("NUMBER_OF_PROCESSORS")
'8'
>>> os.getenv("OS")
'Windows_NT'
>>>
```

Table 4.4 os module: some file-system commands

Function	Description
`os.listdir(path='.')`	List the entries in the directory given by *path* (or the current working directory if this is not specified).
`os.remove(path)`	Delete the file *path* (raises an `OSError` if *path* is a directory; use `os.rmdir` instead).
`os.rename(old_name, new_name)`	Rename the file or directory *old_name* to *new_name*. If a file with the name *new_name* already exists, *it will be overwritten* (subject to user-permissions).
`os.rmdir(path)`	Delete the directory *path*. If the directory is not empty, an `OSError` is raised.
`os.mkdir(path)`	Create the directory named *path*.
`os.system(command)`	Execute *command* in a subshell. If the command generates any output, it is redirected to the interpreter standard output stream, `stdout`.

4.2.2. THE OS MODULE

`os.path` – udostępnia wiele funkcji do manipulowania ścieżkami, nazwami plików lub katalogów itp. Szczególnie przydatne gdy tworzy się program zapisujący wyniki w postaci pliku.

```
>>> os.path.basename("C:\\Users\\kubac\\przyklad\\kwadrat.py")
'kwadrat.py'
>>> os.path.dirname("C:\\Users\\kubac\\przyklad\\kwadrat.py")
'C:\\Users\\kubac\\przyklad'
>>> os.path.split("C:\\Users\\kubac\\przyklad\\kwadrat.py")
('C:\\Users\\kubac\\przyklad', 'kwadrat.py')
>>> os.path.splitext("C:\\Users\\kubac\\przyklad\\kwadrat.py")
('C:\\Users\\kubac\\przyklad\\kwadrat', '.py')
>>> os.path.basename("C:\\Users\\kubac\\przyklad\\kwadrat.py")
'kwadrat.py'
```

Table 4.5 `os.path` module: common pathname manipulations

Function	Description
`os.path.basename(path)`	Return the basename of the pathname *path* giving a relative or absolute path to the file: this usually means the filename.
`os.path.dirname(path)`	Return the directory of the pathname *path*.
`os.path.exists(path)`	Return `True` if the directory or file *path* exists, and `False` otherwise.
`os.path.getmtime(path)`	Return the time of last modification of *path*.
`os.path.getsize(path)`	Return the size of *path* in bytes.
`os.path.join(path1, path2, ...)`	Return a pathname formed by joining the path components *path1*, *path2*, etc. with the directory separator appropriate to the operating system being used.
`os.path.split(path)`	Split *path* into a directory and a filename, returned as a tuple (equivalent to calling `dirname` and `basename`) respectively.
`os.path.splitext(path)`	Split *path* into a "root" and an "extension" (returned as a tuple pair).

4.5. MODULES AND PACKAGES

Możliwości języka Python mogą być w łatwy sposób rozszerzane poza jego podstawowe funkcje przy pomocy *modułów*.

Aby zaimportować dany moduł należy skorzystać ze składni import:

```
import <moduł>
```

Żeby odnieść się do atrybutu znajdującego się w danym module należy wywołać jego nazwę a następnie nazwę atrybutu:

```
<moduł>.atrybut()
```

4.5. MODULES AND PACKAGES

```
zmienna = "wartość"

def funkcja(x):
    y = x**2
    return y


if __name__ == "__main__":
    print(funkcja(10))
    print(zmienna)
    zmienna = "inna wartość"
    print(zmienna)
```

Importując jako moduł

```
>>> import os
>>> os.chdir("C:\\Users\\kubac\\przyklad")
>>> import moduł
>>> moduł.zmienna
'wartość'
>>> moduł.funkcja(25)
625
```

Uruchamiając jako program

```
100
wartość
inna wartość
>>>
```

4.5. MODULES AND PACKAGES

Paczki są sposobem organizacji modułów.

Moduły należące do paczki są przechowywane w katalogu wraz z plikiem `__init__.py` który jest uruchamiany gdy paczka jest importowana.

```
numpy/
    __init__.py
    core/
    fft/
        __init__.py
        fftpack.py
        info.py
        ...
    linalg/
        __init__.py
        linalg.py
        info.py
        ...
    polynomial/
        __init__.py
        chebyshev.py
        hermite.py
        legendre.py
        ...
    random/
    version.py
    ...
```

4.5.1. THE RANDOM MODULE

Oferuje wiele funkcji opartych na *generatorze liczb pseudolosowych*.

```
>>> import random
>>> random.random()
0.025010755222666936
>>> random.seed(42)
>>> random.random()
0.6394267984578837
>>> random.random()
0.025010755222666936
>>> random.seed(42)
>>> random.random()
0.6394267984578837
>>> random.random()
0.025010755222666936
```

```
>>> random.uniform(-2,2)
-0.899882726523523
>>> random.uniform(-2,2)
-1.107157047404709
>>> random.uniform(-2,2)
0.9458848566560496
>>> random.normalvariate(0,1)
2.811460417586422
>>> random.normalvariate(0,1)
-1.2258167276173486
>>> random.normalvariate(0,1)
0.009437580815661357
>>>
```

```
>>> random.randint(5,10)
6
>>> random.randint(5,10)
10
>>> random.randint(5,10)
10
>>> random.randint(5,10)
10
>>> random.randint(5,10)
9
```

```
>>> seq = [10,5,2,'n',-3.4]
>>> random.choice(seq)
'n'
>>> random.choice(seq)
5
>>> random.shuffle(seq)
>>> seq
[5, -3.4, 10, 2, 'n']
>>> random.shuffle(seq)
>>> seq
['n', 5, -3.4, 10, 2]
```

4.5.2. THE URLLIB PACKAGE

Paczka zawierająca wiele modułów pozwalających na otwieranie i otrzymywanie materiałów z adresów internetowych, zazwyczaj przy pomocy protokołu HTTP(S) lub FTP.

```
import urllib.request

req = urllib.request.Request('https://www.wikipedia.org')
response = urllib.request.urlopen(req)
content = response.read()
if (charset := response.headers.get_content_charset()) != None:
    pass
else:
    charset = "utf-8"
html = content.decode(charset)


>>> html[:60]
'<!DOCTYPE html>\n<html lang="en" class="no-js">\n<head>\n<meta '
>>>
```

4.5.3. THE DATETIME MODULE

Udostępnia obiekty przechowujące informacje na temat czasu i dat.

```
>>> from datetime import date
>>> birthday = date(2004,11,5)
>>> today = date.today()
>>> birthday
datetime.date(2004, 11, 5)
>>> today
datetime.date(2021, 4, 11)
>>> birthday.isoformat()
'2004-11-05'
>>> birthday.weekday()
4
>>> birthday.isoweekday()
5
>>> today.ctime()
'Sun Apr 11 00:00:00 2021'
>>> birthday < today
True
>>> today == birthday
False
```

```
>>> from datetime import time
>>> lunchtime = time(hour = 13, minute = 30)
>>> lunchtime
datetime.time(13, 30)
>>> lunchtime.isoformat()
'13:30:00'
>>> precise_time = time(4,46,36,501982)
>>> precise_time.isoformat()
'04:46:36.501982'
>>> witching_hour = time(24)
Traceback (most recent call last):
  File "<pyshell#25>", line 1, in <module>
    witching_hour = time(24)
ValueError: hour must be in 0..23
```

```
>>> from datetime import datetime
>>> now = datetime.now()
>>> now
datetime.datetime(2021, 4, 11, 14, 27, 57, 402478)
>>> now.isoformat()
'2021-04-11T14:27:57.402478'
>>> now.ctime()
'Sun Apr 11 14:27:57 2021'
```

4.6. THE INTRODUCTION TO OBJECT-ORIENTED PROGRAMMING

Do tej pory (w trakcie prezentacji) większość napisanego kodu była rodzajem programowania *proceduralnego*, tzn. były serią ściśle określonych poleceń wykonywanych po kolei.

Innym paradygmatem programowania, popularnym dla większych projektów, jest programowanie *obiektowe* (ang. Object-oriented programming; OOP) – polegające na rozbiciu złożonego problemu na *obiekty* z własnymi metodami i atrybutami oraz ustalaniu relacji między nimi.

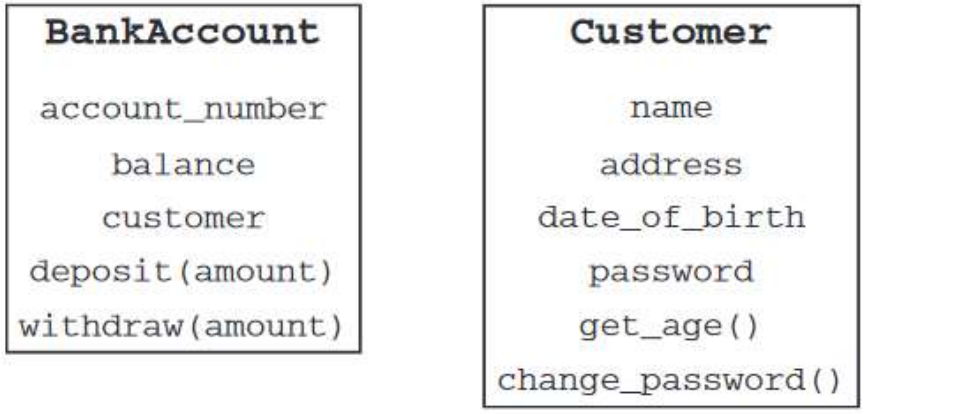


Figure 4.2 Basic classes representing a bank account and a customer.

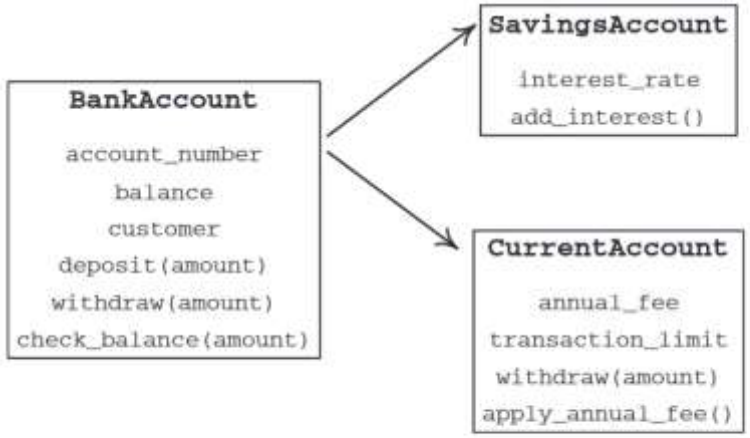


Figure 4.3 Two classes derived from an abstract base class: `SavingsAccount` and `CurrentAccount` *inherit* methods and attributes from `BankAccount` but also customize and extend its functionality.

4.6.1. OBJECT-ORIENTED PROGRAMMING BASICS

Obiekt przechowuje dane o sobie (atrybuty) oraz posiada funkcje (metody) do manipulowania tymi danymi. Obiekt jest utworzony (zainicjowany) ze schematu będącego *klasą*.

```
>>> a = 'good morning'
>>> type(a)
<class 'str'>
>>> a.capitalize()
'Good morning'
>>> a.split()
['good', 'morning']
>>> a * 2
'good morninggood morning'
>>> a.__mul__(2)
'good morninggood morning'
```

```
>>> help(str)
Help on class str in module builtins:

class str(object)
 |  str(object='') -> str
 |  str(bytes_or_buffer[, encoding[, errors]]) -> str
 |
 |  Create a new string object from the given object. If encoding or
 |  errors is specified, then the object must expose a data buffer
 |  that will be decoded using the given encoding and error handler.
 |  Otherwise, returns the result of object.__str__() (if defined)
 |  or repr(object).
 |  encoding defaults to sys.getdefaultencoding().
 |  errors defaults to 'strict'.
 |
 |  Methods defined here:
 |
 |  __mul__(self, value, /)
 |      Return self*value.
 |
 |  capitalize(self, /)
 |      Return a capitalized version of the string.
 |
 |      More specifically, make the first character have upper case and the rest lower
 |      case.
```

4.6.2. DEFINING AND USING CLASSES IN PYTHON

Aby zdefiniować klasę należy użyć słowa kluczowego `class`, definiując metody i atrybuty wewnątrz bloku kodu pod nazwą.

Metody wewnątrz klasy powinny zawierać zmienną `self` która odnosi się do samego obiektu i jego atrybutów/metod.

```
class NazwaKlasy:

    def __init__(self, argument):
        self.argument = argument

    def metoda(self):
        print(self.argument)
```

```python
class BankAccount:
    """Base for representing a bank account"""
    currency = '$'

    def __init__(self, customer, account_number, balance = 0):
        """
        Initialize the BankAccount class with a customer, account number
        and opening balance (which defaults to 0)
        """

        self.customer = customer
        self.account_number = account_number
        self.balance = balance

    def deposit(self, amount):
        """Deposit amount into the bank account"""
        if amount > 0:
            self.balance += amount
        else:
            print("Invalid deposit amount:", amount)

    def withdraw(self, amount):
        """
        Withdraw amount from the bank account, ensuring there are
        sufficient funds.
        """
        if amount > 0:
            if amount > self.balance:
                print("Insufficient funds")
            else:
                self.balance -= amount
        else:
            print("Invalid withdrawal amount:", amount)

    def check_balance(self):
        """ Print a statement of the account balance"""
        print('The balance of the account number {:d} is {:s}{:.2f}'
              .format(self.account_number,self.currency,self.balance))
```

```
>>> my_account = BankAccount('Jakub Olszewski', 2137288)
>>> my_account.account_number
2137288
>>> my_account.balance
0
>>> my_account.customer
'Jakub Olszewski'
>>> my_account.deposit(1000)
>>> my_account.check_balance()
The balance of the account number 2137288 is $1000.00
```

4.6.3. CLASS INHERITANCE IN PYTHON

Odziedziczalność klas jest ich bardzo przydatną własnością, pozwalającą na napisanie uogólnionych schematów obiektów a następnie tworzeniu innych wyspecjalizowanych wersji w oparciu o te schematy.

Aby stworzyć podklasę, definiując ją obok nazwy w nawiasie () należy podać nazwę klasy nadrzędnej:

```
class NazwaKlasy:

    def __init__(self, argument):
        self.argument = argument

    def metoda(self):
        print(self.argument)


class PodKlasa(NazwaKlasy):

    def __init__(self,argument, jakiś_inny_argument):
        self.jakiś_inny_argument = jakiś_inny_argument
        super().__init__(argument)
```

```python
class SavingsAccount(BankAccount):
    """A class representing a savings account."""

    def __init__(self, customer, account_number, interest_rate, balance = 0):
        """ Initialize the savings account. """
        self.interest_rate = interest_rate
        super().__init__(customer, account_number, balance)

    def add_interest(self):
        """ Add interest to the account at the rate self.interest_rate."""

        self.balance *= (1. + self.interest_rate / 100)
```

```python
def __init__(self, customer, account_number, balance = 0):
    """
    Initialize the BankAccount class with a customer, account number
    and opening balance (which defaults to 0)
    """

    self.customer = customer
    self.account_number = account_number
    self.balance = balance
```

```
>>> save_account = SavingsAccount("Jakub Olszewski", 2137288, 5.5, 1000)
>>> save_account.check_balance()
The balance of the account number 2137288 is $1000.00
>>> save_account.add_interest()
>>> save_account.check_balance()
The balance of the account number 2137288 is $1055.00
```

4.6.4. CLASSES AND OPERATORS

Operatory (jak +, * lub <=) oraz funkcje jak `len` czy `abs` działają na obiektach wywołując metody zdefiniowane wewnątrz tych obiektów których nazwy zaczynają się oraz kończą dwoma znakami podkreślenia `__` (tzw. „dunder" methods). Aby obiekt definiowany przez użytkownika obsługiwał takie metody wystarczy zdefiniować te funkcje wewnątrz klasy.

```
>>> x = 10
>>> y = 20
>>> x + y
30
>>> x.__add__(y)
30
```

```
>>> save_account + 10
Traceback (most recent call last):
  File "<pyshell#32>", line 1, in <module>
    save_account + 10
TypeError: unsupported operand type(s) for +: 'SavingsAccount' and 'int'
```

Table 4.8 Common Python special methods

Method	Description	Example
`__add__`	+, addition	`x + y`
`__sub__`	-, subtraction	`x - y`
`__mul__`	*, multiplication	`x * y`
`__truediv__`	/, "true" division	`x / y`
`__floordiv__`	//, floor division	`x // y`
`__mod__`	%, modulus	`x % y`
`__pow__`	**, exponentiation	`x ** y`
`__neg__`	negation (unary minus)	`-x`
`__matmul__`	@, matrix multiplication	`x @ y`
`__abs__`	absolute value	`abs(x)`
`__contains__`	membership	`y in x`
`__lt__`	less than	`y < x`
`__le__`	less than or equal to	`y <= x`
`__eq__`	equal to	`y == x`
`__ne__`	not equal to†	`y != x`
`__gt__`	greater than	`y > x`
`__ge__`	greater than or equal to	`y >= x`
`__str__`	human-readable string representation	`str(x)`
`__repr__`	unambiguous string representation	`repr(x)`

† If not explicitly implemented, `__ne__` calls `__eq__` and inverts the result.

```python
class Vector2D:
    """ A two-dimensional vector with Cartesian coordinates."""

    def __init__(self, x, y):
        self.x, self.y = x,y

    def __str__(self):
        """Human-readable string representation of the vector."""
        return '{:g}i + {:g}j'.format(self.x, self.y)

    def __repr__(self):
        """Unambiguous string representation of the vector."""
        return repr((self.x, self.y))

    def dot(self, other):
        """The scalar (dot) product of self and other. Both must be vectors."""
        if not isinstance(other, Vector2D):
            raise TypeError("Can only take dot product of two Vector2D objects")
        return self.x * other.x + self.y * other.y

    __matmul__ = dot
```

```
>>> x = Vector2D(10,20)
>>> y = Vector2D(14,34)
>>> x.dot(y)
820
>>> x @ y
820
>>> str(x)
'10i + 20j'
>>> repr(x)
'(10, 20)'
```