

Setup & Tools

IDE (Visual Studio Code z wtyczką do Pythona)

1. Instalacja Visual Studio Code
2. Instalacja wtyczki do Pythona
3. Otwarcie katalogu z repozytorium
4. Opis interfejsu - pięć paneli bocznych
5. F1 - znajdź polecenie
6. ctrl+P - przełączaj między plikami
7. Wsparcie dla git (dolny paszek)
8. Pullowanie repozytorium
9. Dolny panel uaktywniany ctrl+ `

Instalacja Pythona 2.7

Python 2.7 powinien być zainstalowany pod Debianem.

Pod Windowsem konieczna jest zainstalowanie Pythona. Instalator można pobrać z oficjalnej strony Pythona.

Dobrze jest się upewnić, czy pracujemy na Pythonie 2, a nie 3:

```
python --version  
pip --version
```

Uruchamianie skryptów z terminala

Uruchomienie skryptu:

```
python moj-skrypt.py
```

Uruchomienie interaktywnej konsoli Pythona (REPL):

```
python
```

Instalacja bibliotek za pomocą managera pakietów pip

Instalacja biblioteki dostępnej na pypi:

```
pip install biblioteka
```

Wylistowanie wszystkich zainstalowanych bibliotek:

```
pip freeze
```

Przydatny idiom:

```
pip freeze | grep jakasbiblioteka
```

Interaktywna konsola IPython

Instalacja:

```
pip install ipython
```

Sprawdzenie instalacji:

```
ipython --version
```

Argumenty są takie same jak dla polecenia python.

Uruchomienie interaktywnej konsoli:

```
ipython
```

Uruchomienie skryptu:

```
ipython moj-skrypt.py
```

Wykonanie skryptu, a następnie wejście w interaktywną konsole (REPL):

```
ipython -i moj-skrypt.py
```

We wnętrzu IPyttona

Po uruchomieniu IPythona, można wykonać polecenia basha bez wychodzenia z IPythona, wystarczy poprzedzić je wykrzyknikiem:

```
!python --version
```

Python 2.7.13

Jupyter Notebook to nakładka graficzna na IPythona.

Jupyter Notebook (opcjonalnie)

Instalacja:

```
pip install jupyter
```

Przed uruchomieniem należy przejść do katalogu, w którym chcemy pracować.

Uruchomienie:

```
jupyter notebook
```

Powinna wyświetlić się nowa zakładka w domyślnej przeglądarce.

Można przeglądać poszczególne pliki z katalogu, który wybrałeś, oraz z podkatalogów. Można także otworzyć poszczególne notebooki i zacząć je edytować.

Aby wyjść z Jupyter Notebooka, nie wystarczy zamknąć zakładki w przeglądarce. Należy także wyłączyć serwer - w konsoli zatrzymujemy go klikając **ctrl+C** i potwierdzając **y**.

Dobrze jest usunąć ewentualne zmiany wprowadzone w notebookach, tak aby przy pullowaniu nie doszło do konfliktu.

Builtin Data Structures

Liczby (float i int)

```
1
```

```
1
```

```
1.0
```

```
1.0
```

```
2**100
```

```
1267650600228229401496703205376
```

```
1 / 2
```

```
0.5
```

```
1.0 / 2
```

```
0.5
```

```
x = 1  
y = 2  
x / float(y)
```

0.5

```
print(2/3)
from __future__ import division
print(2/3)
```

```
0
0.666666666667
```

```
import sys
```

```
sys.version
```

```
'2.7.10 (default, Aug 17 2018, 19:45:58) \n[GCC 4.2.1 Compatible Apple LLVM 10.0.0 (clang-1000.0.42)]'
```

```
import sys
if sys.version_info.major != 3:
    print("Zla wersja")
    # sys.exit(1)
```

```
Zla wersja
```

String

```
'asdf'
```

```
'asdf'
```

```
print("qwer\nasdf")
```

```
qwer
asdf
```

```
print(r"qwer\nasdf")
```

```
qwer\nasdf
```

String Formatting

```
"%s - %d - %.3f" % ('asdf', 2, 2.34556789)
```

```
'asdf - 2 - 2.346'
```

```
{1} {0:.3f}'.format(2.34567, 'asdf')
```

```
'asdf 2.346'
```

```
p = '{} {}'
```

```
a = 2
```

```
b = 'asdf'
```

```
p.format(a, b)
```

```
'2 asdf'
```

```
pattern = 'Name: {}, family name: {}'
pattern.format('John', 'Smith')
```

```
'Name: John, family name: Smith'
```

```
x = 2
y = 3.345678
z = x + y
print("x to {}, y to {:.3f}")
```

```
x to 2, y to 5.346
```

Użyteczne metody

```
s = "asdf qwer\nzxcv hjkl"
```

```
s2 = s.upper()
```

```
s
```

```
'asdf qwer\nzxcv hjk!'
```

```
s2
```

```
'ASDF QWER\nZXCV HJKL'
```

```
s.split()
```

```
['asdf', 'qwer', 'zxcv', 'hjk!']
```

```
s.split(' ')
```

```
['asdf', 'qwer\nzxcv', '', 'hjk!']
```

```
s.split(None, 2)
```

```
['asdf', 'qwer', 'zxcv hjk!']
```

```
s.rsplit(None, 2)
```

```
['asdf qwer', 'zxcv', 'hjk!']
```

```
s.find('qwer')
```

```
5
```

```
s[5:]
```

```
'qwer\nzxcv hjk!'
```

```
len(s)
```

```
20
```

```
', '.join(['asdf', 'qwer', 'asdf zxcv'])
```

```
'asdf, qwer, asdf zxcv'
```

List (lista)

```
l = ['a', 'b', 'c']
```

```
for elem in l:  
    print(elem)
```

```
a  
b  
c
```

```
for i, elem in enumerate(l):  
    print(i, elem)
```

```
0 a  
1 b  
2 c  
3 d
```

```
for i in range(5):  
    print(i)
```

```
0  
1  
2  
3  
4
```

```
l.append('d')  
l
```

```
['a', 'b', 'c', 'd']
```

```
len(l)
```

```
4
```

```
l[0]
```

```
'a'
```

☆ WC

Napisz program, który działa podobnie do polecenia wc -- zwraca liczbę linii, słów i znaków w danym pliku. Nazwa pliku jest przekazana jako argument.

Kod początkowy

```
#!/usr/bin/env python
import sys

filename = sys.argv[1]
with open(filename) as s:
    # Twój kod
    # for po s
    print(lines, words, characters)
```

Podpowiedź

```
#!/usr/bin/env python
import sys

filename = sys.argv[1]
with open(filename) as s:
    # Z użyciem pętli for
    lines = 0
    words = 0
    characters = 0
    for line in s:
        lines += ...
        words += ...
        characters += ...
    print(lines, words, characters)
```

Rozwiązanie

```
#!/usr/bin/env python
import sys

filename = sys.argv[1]
with open(filename) as s:
    lines = 0
    words = 0
    characters = 0
    for line in s:
        lines += 1
        words += len(line.split())
        characters += len(line)
    print(lines, words, characters)
```

```
## Wersja z wieloma plikami:
#!/usr/bin/env python
import sys

for filename in sys.argv[1:]:
    with open(filename) as s:
        lines = 0
        words = 0
        characters = 0
        for line in s:
            lines += 1
            words += len(line.split())
            characters += len(line)
        print(filename, lines, words, characters)
```

Wyrażenia listowe (list comprehension)

```
original = ['a', 'b', 'c']
new = []
for char in original:
    new.append(char.upper())
```

```
    new.append(new_elem)
new
```

```
['A', 'B', 'C']
```

```
new = [char.upper() for char in original]
new
```

```
['A', 'B', 'C']
```

☆ Suma cyfr 2**1000

Znajdź sumę cyfr liczby 2^{1000} .

Rozwiąż to zadanie na kilka sposobów:

1. używając pętli for, ale nie używając wyrażeń generatorowych
2. używając wyrażenia listowego i funkcji sum,

Poprawny wynik: 1366.

```
2**1000
```

```
107150860718626732094842504906000181056140481170553360744375038837035105112493612249319837881569585812759467291  
755314682518714528569231404359845775746985748039345677748242309854210746050623711418779541821530464749835819412  
67398767559165543946077062914571196477686542167660429831652624386837205668069376
```

```
str(1234)
```

```
'1234'
```

```
int('1234')
```

```
1234
```

```
sum([2, 3, 4])
```

```
9
```

```
for char in 'asdf':  
    print(char)
```

```
a  
s  
d  
f
```

```
s = 'asdf'  
s[1]
```

```
's'
```

Rozwiązanie - pętla for

```
counter = 0
number = 2**1000
for digit in str(number):
    counter += int(digit)
print(counter)
```

```
counter = 0
for digit in str(2**1000):
    counter += int(digit)
print(counter)
```

```
1366
```

Rozwiązanie - wyrażenie listowe

```
sum( [int(digit) for digit in str(2**1000)] )
```

```
1366
```

☆ Wypisz sys.argv

Napisz narzędzie konsolowe (skrypt) print-args.py, który wypisuje wszystkie argumenty przekazane do niego, wielkimi literami.

Kod początkowy

```
#!/usr/bin/env python
import sys

if __name__ == "__main__":
    print(sys.argv)
```

Rozwiązanie

```
#!/usr/bin/env python
import sys

if __name__ == "__main__":
    # pierwszy sposób
    print([arg.upper() for arg in sys.argv[1:]])

    # drugi sposób
    for arg in sys.argv[1:]:
        print(arg.upper())
```

Oczekiwane zachowanie

```
!python print-args.py
```

```
[]
```

```
!python print-args.py arg1 arg2 3 "4 4 4"
```

```
[ARG1, 'ARG2', '3', '4 4 4']
```

```
!chmod 700 print-args.py
./print-args.py
```

```
[]
```

Zaawansowane wyrażenia listowe

```
range(5)
```

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

```
old = [0, 1, 2, 3, 4]
new = []
for i in old:
    if i % 2 == 0:
        new.append(i)
new
```

```
[0, 2, 4]
```

```
[i for i in old if i % 2 == 0]
```

```
[0, 2, 4]
```

```
[i for i in range(5)
if i % 2 == 0 and i % 3 != 0]
```

```
[2, 4]
```

☆ Suma liczb podzielnych przez 3 lub 5

Znajdź sumę wszystkich liczb całkowitych od 0 do 1000, które są podzielne przez 3 lub 5.

Rozwiąż to zadanie na kilka sposobów:

1. używając pętli for, ale nie używając wyrażeń listowych,
2. używając wyrażenia listowych i funkcji sum,

Poprawny wynik: 234168.

Rozwiążanie - pętla for

```
s = 0
for i in range(1001):
    if i % 5 == 0 or i % 3 == 0:
        s += i
print(s)
```

234168

Rozwiążanie - wyrażenie listowe

```
sum([i for i in range(1001)
      if i % 5 == 0 or i % 3 == 0])
```

234168

Krotka (tuple)

```
x = ('a', 2)
```

x

('a', 2)

```
x[0]
```

'a'

```
('a',)
```

('a',)

```
'a',
```

('a',)

```
t = 'a',
```

t

('a',)

```
()
```

0

```
l = ['a', 'b', 'c']
l[1] = 'BBB'
l
```

['a', 'BBB', 'c']

```
t = ('a', 'b', 'c')
t[1] = 'BBB'
```

```
-----  
TypeError                      Traceback (most recent call last)
<ipython-input-56-b44b50bf4dc5> in <module>()
      1 t = ('a', 'b', 'c')
----> 2 t[1] = 'BBB'
```

TypeError: 'tuple' object does not support item assignment

```
s = "abc"
s[1] = "B"
s
```

```
-----  
TypeError                      Traceback (most recent call last)
<ipython-input-57-fe50ed1fad13> in <module>()
      1 s = "abc"
----> 2 s[1] = "B"
      3 s
```

```
TypeError: 'str' object does not support item assignment
```

```
1 + " sek"
```

```
-----  
TypeError          Traceback (most recent call last)  
<ipython-input-60-4b4190eb4e43> in <module>()  
----> 1 1 + " sek"
```

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

```
x = 1  
f'{x} sek"
```

```
'1 sek'
```

```
1 + int("2")
```

```
3
```

Tuple Packing and Unpacking

```
t = ('aaaa', 'bbbb')
```

```
a, b = t
```

```
a
```

```
'aaaa'
```

```
b
```

```
'bbbb'
```

```
t
```

```
('aaaa', 'bbbb')
```

☆ swap

```
a = 'aaaa'  
b = 'bbbb'
```

```
# Twoje rozwiązańe  
# Pierwszy sposób: w trzech liniach, z jedną zmienną pomocniczą  
# Drugi sposób: w jednej linii, z tuple unpacking, bez zmiennych pomocniczych
```

```
print(a) # ==> bbbb  
print(b) # ==> aaaa
```

```
bbbb  
aaaa
```

```
a = 'aaaa'  
b = 'bbbb'
```

```
t = a, b  
b, a = t
```

```
print(a) # ==> bbbb  
print(b) # ==> aaaa
```

```
bbbb  
aaaa
```

```
a = 'aaaa'  
b = 'bbbb'
```

```
a, b = b, a
```

```
print(a) # ==> bbbb  
print(b) # ==> aaaa
```

```
bbbb  
aaaa
```

```
a = 'aaaa'  
b = 'bbbb'  
  
c = a  
a = b  
b = c  
  
print(a) # ==> bbbb  
print(b) # ==> aaaa
```

```
bbbb  
aaaa
```

```
a = 0  
b = 1  
for i in range(10):  
    print(a)  
  
    # c = a+b  
    # a = b  
    # b = c  
  
    a, b = a+b, a
```

```
0  
1  
1  
2  
3  
5  
8  
13  
21  
34
```

Struktury zagnieżdżone

☆ Struktury zagnieżdżone

Stwórz listę osób. Każda osoba jest reprezentowana przez krotkę składającą się imienia i nazwiska tej osoby. Następnie wyświetl w kolejnych liniach nazwiska tych osób.

Rozwiązanie

```
people = [  
    ('john', 'smith'),  
    ('bob', 'wilson'),  
    ('alice', 'smith'),  
]  
for _, family in people:  
    print(family)
```

```
smith  
wilson  
smith
```

Słownik (dict)

```
d = {'key': 'value'}
```

```
d = {  
    'key': 'value',  
    'key2': 'value2',  
}
```

```
d
```

```
{'key': 'value', 'key2': 'value2'}
```

```
d['new-key'] = 52
```

```
d
```

```
{'key': 'value', 'key2': 'value2', 'new-key': 52}
```

```
d['key']
```

```
'value'
```

```
d['non-existing'] # KeyError
```

```
KeyError Traceback (most recent call last)
<ipython-input-95-1e92e99e4857> in <module>()
----> 1 d['non-existing'] # KeyError
```

```
KeyError: 'non-existing'
```

```
'non-existing' in d
```

```
False
```

```
'non-existing' not in d
```

```
True
```

```
for k in d: # iteracja po kluczach
    print(k)
```

```
key
key2
new-key
```

```
for v in d.values(): # iteracja po wartościach
    print(v)
```

```
value
value2
52
```

```
for k, v in d.items(): # iteracja i po kluczach i po wartościach
    print(k)
    print(v)
```

```
key
value
key2
value2
new-key
52
```

☆ zliczanie słów

Zlicz jak często występują słowa w zadanym napisie. Wielkość liter nie powinna mieć znaczenia.

Kod początkowy

```
text = 'Ala ma kota ala ala ala'
```

Rozwiązanie

```
words = text.lower().split()
counter = {}
for word in words:
    if word not in counter:
        counter[word] = 0
    counter[word] += 1

for word, frequency in counter.items():
    print('{} -> {}'.format(word, frequency))
```

```
ala -> 4
ma -> 1
kota -> 1
```

Dictionary Comprehension

```
{i: bin(i) for i in range(5)}  
{0: '0b0', 1: '0b1', 2: '0b10', 3: '0b11', 4: '0b100'}
```

```
{i: bin(i) for i in range(5) if i % 2 == 0}  
{0: '0b0', 2: '0b10', 4: '0b100'}
```

```
I = [[1, 2, 3], [4, 6], [7, 8, 9]]
```

```
[elem  
for row in I  
if len(row) == 3  
for elem in row  
if elem % 2 == 0]
```

```
[2, 8]
```

```
new = []  
for row in I:  
    if len(row) == 3:  
        for elem in row:  
            if elem % 2 == 0:  
                new.append(elem)  
new
```

```
[2, 8]
```

Control Flow

Instrukcje warunkowe

```
if 2 == 3:  
    print("True")  
else:  
    print("False")
```

```
False
```

```
a = 3  
if a % 3 == 0:  
    print("Podzielna")  
elif a % 3 == 1:  
    print("Reszta z dzielenia 1")  
elif a % 3 == 2:  
    print("Reszta z dzielenia 2")
```

```
Podzielna
```

```
a = 3  
if a % 3 == 0:  
    print("Podzielna")  
elif a % 3 == 1:  
    print("Reszta z dzielenia 1")  
else:  
    print("Reszta z dzielenia 2")
```

```
Podzielna
```

```
if 2 == 2:  
    print("A")  
elif 3 == 3:  
    print("B")  
else:  
    print("C")
```

```
A
```

Wyrażenia logiczne

```
True or False
```

```
True
```

```
True and False
```

```
False
```

```
not False
```

```
True
```

```
True == 1
```

```
True
```

```
False == 0
```

```
True
```

```
2 == 3
```

```
False
```

```
not 2 == 3
```

```
True
```

```
2 != 3
```

```
True
```

```
a = True  
b = False  
c = False  
d = True  
(a and b) or (c and d)
```

```
False
```

```
(a or b) and (c or d)
```

```
True
```

```
2 is None
```

```
False
```

```
2 is not None
```

```
True
```

```
a = [1, 2, 3]  
b = [1, 2, 3]  
c = b  
print(a == b)  
print(a is b)  
print(b is c)
```

```
True
```

```
False
```

```
True
```

```
c = b[:]  
c
```

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

```
import copy
```

```
c = copy.copy(b)  
c2 = copy.deepcopy(b)
```

Pętla for i iteracja po kolekcjach

```
lista = [1, 2, 3]  
krotka = (1, 2, 3)  
slownik = {'key': 'value', 'key2': 'value2'}  
zbior = {1, 2, 3}
```

```
for elem in lista:  
    print(elem)
```

```
1  
2  
3
```

```
for elem in krotka:  
    print(elem)
```

```
1  
2  
3
```

```
for elem in slownik:  
    print(elem)
```

```
key  
key2
```

```
for key, value in slownik.items():  
    print(f'{key} -> {value}')
```

```
key -> value  
key2 -> value2
```

```
for elem in zbior:  
    print(elem)
```

```
1  
2  
3
```

```
l = [1, 2, 3]  
l2 = ['john', 'bob', 'alice']  
print(l + l2)  
print(l)
```

```
[1, 2, 3, 'john', 'bob', 'alice']  
[1, 2, 3]
```

```
'john' + 'qwer'
```

```
'johnqwer'
```

```
l = [1, 2, 3]  
l2 = ['john', 'bob', 'alice']  
l.append(l2)  
l
```

```
[1, 2, 3, ['john', 'bob', 'alice']]
```

```
l = [1, 2, 3]  
l2 = ['john', 'bob', 'alice']  
l.extend(l2)  
l
```

```
[1, 2, 3, 'john', 'bob', 'alice']
```

```
l = [1, 2, 'john', 'bob']  
[elem for elem in l if isinstance(elem, str)]
```

```
['john', 'bob']
```

Pętla while

```
licznik = 5  
while licznik > 0:  
    print(licznik)  
    licznik -= 1  
print("Koniec")
```

```
5  
4  
3  
2  
1  
Koniec
```

```
licznik = 6  
while True:
```

```
while True:  
    licznik := 1  
    if licznik == 0:  
        break  
    print(licznik)
```

5
4
3
2
1

break, continue w pętlach

```
licznik = 5  
while licznik > 0: # punkt B  
    if licznik == 3:  
        licznik -= 1  
        continue # natychmiastowe przejście do kolejnej iteracji, czyli do punktu B  
    if licznik == 1:  
        break # natychmiastowe wyjście z pętli, czyli przejdzie do punktu A  
    print(licznik)  
    licznik -= 1  
# punkt A  
print("Koniec")
```

5
4
2
Koniec

Wyjątki i obsługa błędów

Podstawy

```
print('start')  
x = 1 / 0  
print('finish')  
  
start  
  
-----  
ZeroDivisionError      Traceback (most recent call last)  
<ipython-input-15-ef0cf86e9deb> in <module>()  
      1 print('start')  
----> 2 1 / 0  
      3 print('finish')
```

ZeroDivisionError: integer division or modulo by zero

```
try:  
    print('start try')  
    1 / 0  
    print('finish try')  
except ZeroDivisionError:  
    print('except')  
print('out')
```

start try
except
out

```
try:  
    print('start try')  
    1 / 2  
    print('finish try')  
except ZeroDivisionError:  
    print('except')  
print('out')
```

start try
finish try
out

```
try:  
    print('start try')  
    1 / 0 # ZeroDivisionError
```

```
print('finish try')
except KeyError:
    print('except')
print('out')
```

start try

```
-----  
ZeroDivisionError      Traceback (most recent call last)  
<ipython-input-38-7a940178abe8> in <module>()  
  1 try:  
  2     print('start try')  
----> 3     1 / 0 # ZeroDivisionError  
  4     print('finish try')  
  5 except KeyError:
```

ZeroDivisionError: integer division or modulo by zero

Zaawansowane użycie

```
# try executing the following statements;  
# on any error, abort and go to appropriate  
# except clause if there is any  
try:  
    print('start try')  
    x = 1 / 0  
    print('finish try')  
except ZeroDivisionError: # executed only when you try to divide by zero in try clause  
    print('except ZeroDivisionError')  
except ValueError: # executed only if ValueError is raised in try clause  
    print('except ValueError')  
else: # executed only on success; exceptions are not caught  
    print('start else', x)  
    y = x / 0 # yet another error  
    print('finish else')  
finally: # always executed  
    print('finally')
```

start try
except ZeroDivisionError
finally

```
try:  
    print('start try')  
    x = 1 / 2  
    print('finish try')  
except ZeroDivisionError:  
    print('except KeyError')  
except ValueError:  
    print('except ValueError')  
else:  
    print('start else', x)  
    y = x / 2  
    print('finish else')  
finally:  
    print('finally')
```

start try
finish try
start else 0.5
finish else
finally

```
try:  
    print('start try')  
    x = 1 / 2  
    print('finish try')  
except ZeroDivisionError:  
    print('except KeyError')  
except ValueError:  
    print('except ValueError')  
else:  
    print('start else', x)  
    y = x / 0 # yet another error  
    print('finish else')  
finally:  
    print('finally')  
print('out')
```

```
start try
finish try
start else 0.5
finally
```

```
ZeroDivisionError          Traceback (most recent call last)
<ipython-input-153-af39575d91d3> in <module>()
      9 else:
     10     print('start else', x)
--> 11     y = x / 0 # yet another error
     12     print('finish else')
     13 finally:
```

```
ZeroDivisionError: float division by zero
```

```
try:
    print('start try')
    x = 1 / 2
    print('finish try')
except ZeroDivisionError:
    print('except ZeroDivisionError')
except ValueError:
    print('except ValueError')
else:
    print('start else', x)
    y = x / 0 # yet another error
    print('finish else')
finally:
    print('finally')
print('out')
```

```
start try
finish try
start else 0.5
finally
```

```
ZeroDivisionError          Traceback (most recent call last)
<ipython-input-154-af39575d91d3> in <module>()
      9 else:
     10     print('start else', x)
--> 11     y = x / 0 # yet another error
     12     print('finish else')
     13 finally:
```

```
ZeroDivisionError: float division by zero
```

```
try:
    print('start try')
    x = 1 / 2
    print('finish try')
    print('start else', x)
    y = x / 0 # yet another error
    print('finish else')
except ZeroDivisionError:
    print('except ZeroDivisionError')
except ValueError:
    print('except ValueError')
finally:
    print('finally')
print('out')
```

```
start try
finish try
start else 0.5
except KeyError
finally
out
```

Raising Exceptions

```
raise ValueError
```

```
ValueError          Traceback (most recent call last)
<ipython-input-44-e4c8e09828d5> in <module>()
----> 1 raise ValueError
```

```
ValueError:
```

```
raise ValueError('argument must be positive integer')
```

```
-----  
ValueError          Traceback (most recent call last)  
<ipython-input-45-55f75a23829a> in <module>()  
----> 1 raise ValueError('argument must be positive integer')
```

ValueError: argument must be positive integer

Ask for Permission vs Ask for Forgiveness

```
I = ['a', 'b']  
if len(I) >= 3:  
    third = I[2]  
else:  
    third = None  
print(third)
```

None

```
I = ['a', 'b']  
try:  
    third = I[2]  
except IndexError:  
    third = None  
print(third)
```

None

☆ Dzielenie liczb

```
x = input('Pierwsza liczba: ')
```

Pierwsza liczba: 2

```
x
```

```
'2'
```

```
### Rozwiązanie  
x = int(input("Pierwsza liczba: "))  
y = int(input("Druga liczba: "))  
try:  
    result = x / y  
except ZeroDivisionError:  
    print("Dzielenie przez zero")  
else:  
    print(f"Wynik: {result}")
```

Pierwsza liczba: 2

Druga liczba: 2

Wynik: 1.0

```
### Rozwiązanie z if  
x = int(input("Pierwsza liczba: "))  
y = int(input("Druga liczba: "))  
if y != 0:  
    result = x / y  
    print(f"Wynik: {result}")  
else:  
    print("Dzielenie przez zero")
```

Pierwsza liczba: 2

Druga liczba: 1

Wynik: 2.0

Asercje

```
assert 2 == 3
```

```
-----  
AssertionError          Traceback (most recent call last)  
<ipython-input-171-65570b070e9a> in <module>()  
----> 1 assert 2 == 3
```

AssertionError:

```
assert 2 == 3, 'something went wrong'
```

```
AssertionError          Traceback (most recent call last)
<ipython-input-49-02487a584844> in <module>()
----> 1 assert False, 'something went wrong'
```

AssertionError: something went wrong

```
if not 2 == 3:
    raise AssertionError('something went wrong')
```

```
AssertionError          Traceback (most recent call last)
<ipython-input-52-cab995c0d16a> in <module>()
      1 if 2 != 3:
----> 2     raise AssertionError('something went wrong')
```

AssertionError: something went wrong

```
x = 2
assert x == 3, x
```

```
AssertionError          Traceback (most recent call last)
<ipython-input-173-d80f9ca97e54> in <module>()
      1 x = 2
----> 2 assert x == 3, x
```

AssertionError: 2

Menadżery kontekstu (with)

```
s = open('file.txt', 'w')
s.write('asdf')
s.close()
```

```
s = open('file.txt', 'w')
try:
    s.write('asdf')
finally:
    s.close()
```

```
# ekwiwalent tego, co wyżej
with open('file.txt', 'w') as s:
    s.write('asdf')
```

```
try:
    s = open('file.txt', 'w')
except FileNotFoundError:
    print("Brak pliku")
else:
    try:
        s.write('asdf')
    finally:
        s.close()
```

```
try:
    s = open('file.txt', 'w')
except FileNotFoundError:
    print("Brak pliku")
else:
    with s:
        s.write('asdf')
```

```
!rm file.txt # cleanup
```

```
try:
    raise TypeError
except (ValueError, TypeError) as e:
    print(type(e))
```

```
<type 'exceptions.TypeError'>
```

```
# Pułapka!
try:
    raise TypeError
except ValueError, TypeError:
#except ValueError as TypeError:
    print("blad")
```

blad

```
# Poprawnie
try:
    raise TypeError
except (ValueError, TypeError):
    print("blad")
```

blad

☆ REPL

Napisz program działający na zasadzie pytanie - odpowiedź. Taki skrypt składa się z REPL - read, eval, print loop.

Do pobrania jednej linii ze standardowego wejścia użyj funkcji input.

Program powinien wykrywać wcisnięcie Ctrl+C (KeyboardInterrupt) i w takiej sytuacji wyświetlić pożegnanie i zakończyć się.

Dostępne powinny być następujące polecenia (case-insensitive):

- eval - wylicza podane wyrażenie Pythona przy pomocy wbudowanej funkcji "eval". Jeżeli wystąpi wyjątek, należy go obsłużyć i wyświetlić klasę wyjątku.
- exit - wyświetla pożegnanie i kończy działanie programu
- można nie wpisać nic, wówczas ponownie prosimy użytkownika o wpisanie czegoś.
- w przypadku wpisania innego polecenia, wówczas należy wyświetlić "Invalid command" i kontynuować.

Hinty

```
eval("2+2")
```

4

```
input(">>> ")
```

>>> asdf

'asdf'

```
e = KeyError()
type(e).__name__
```

'KeyError'

Rozwiążanie

```
while True:
    expr = input(">>> ")
    result = eval(expr)
    print(result)
```

>>> 2 + 2

4

>>> 2*2

4

```
-----
KeyboardInterrupt          Traceback (most recent call last)
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/ipykernel/kernelbase.py in _input_request(self, prompt, id, parent, password)
```

```
728      try:
--> 729          ident, reply = self.session.recv(self.stdin_socket, 0)
730      except Exception:
```

```
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/jupyter_client/session.py in recv(self, socket, mode, content, copy)
```

```
802      try:
--> 803          msg_list = socket.recv_multipart(mode, copy=copy)
804      except zmq.ZMQError as e:
```

```
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/zmq/sugar/socket.py in recv_multipart(self, flags, copy, tra
ck)
 465      ....
--> 466      parts = [self.recv(flags, copy=copy, track=track)]
 467      # have first part already, only loop while more to receive

zmq/backend/cython/socket.pyx in zmq.backend.cython.socket.Socket.recv()
zmq/backend/cython/socket.pyx in zmq.backend.cython.socket.Socket.recv()
zmq/backend/cython/socket.pyx in zmq.backend.cython.socket._recv_copy()

/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/zmq/backend/cython/checkrc.pxd in zmq.backend.cython.
checkrc._check_rc()
```

KeyboardInterrupt:

During handling of the above exception, another exception occurred:

```
KeyboardInterrupt          Traceback (most recent call last)
<ipython-input-176-31082b07d2cd> in <module>()
 1 while True:
--> 2     expr = input("=>> ")
 3     result = eval(expr)
 4     print(result)
```

```
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/ipykernel/kernelbase.py in raw_input(self, prompt)
 702         self._parent_ident,
 703         self._parent_header,
--> 704         password=False,
 705     )
 706
```

```
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/ipykernel/kernelbase.py in _input_request(self, prompt, id
ent, parent, password)
 732     except KeyboardInterrupt:
 733         # re-raise KeyboardInterrupt, to truncate traceback
--> 734         raise KeyboardInterrupt
 735     else:
 736         break
```

KeyboardInterrupt:

```
### Rozwiązywanie częściowe
try:
    while True:
        # Read
        expr = input("=>> ")

        # Eval
        try:
            result = eval(expr)
        except Exception:
            result = 'Blad.'

        # Print
        print(result)
except KeyboardInterrupt:
    pass
finally:
    print("Bye")
```

```
>>> 2+2
4
>>> 2/0
Blad.
Bye
```

```
def repl():
    try:
        while True:
            line = input('=>> ')
            tokens = line.split(None, 1)

            try:
                command = tokens[0].lower()
            except IndexError:
                command = "
```

```

try:
    rest = tokens[1]
except IndexError:
    rest = ""

if command == 'eval':
    try:
        result = eval(rest)
    except Exception as e:
        print(type(e).__name__)
    else:
        print(result)
elif command == 'exit':
    raise KeyboardInterrupt
elif command == "":
    pass
else:
    print('Invalid command.')
except KeyboardInterrupt:
    pass
finally:
    print('Bye')

```

Oczekiwane zachowanie

```

repl()
>>> eval 2+2
4
>>> eval 2/0
ZeroDivisionError
>>> eval 'asdf + qwer'
asdfqwer
>>>
>>> asdf
Invalid command.
Bye
>>> exit

```

Functions

Definicja i wywołanie

```

def add(a, b):
    return a + b

```

```

suma = add(2, 3)
suma

```

5

```

add(2, 3)

```

5

Parametry pozycyjne i nazwane

```

s = "asdf qwer zxcv"
s.split(maxsplit=1)
['asdf', 'qwer zxcv']

```

☆ Parametry pozycyjne (positional arguments)

Napisz dwuargumentową funkcję, która zwraca zarówno różnicę ($a-b$) jak i iloczyn $a \cdot b$ parametrów. Wywołaj funkcję i wypisz obie wartości w osobnych liniach.

Rozwiązanie

```

def compute(a, b):
    return a-b, a*b

```

```
return a, b
```

```
s = compute(2, 3)
print(s)
print(s[0])
roznica, _ = s
print(roznica)

roznica, iloczyn = compute(2, 3)
print(roznica)
print(iloczyn)
```

```
(-1, 6)
-1
-1
-1
6
```

```
compute(b=3, a=2)
```

```
(-1, 6)
```

```
compute(2, b=3)
```

```
(-1, 6)
```

Wartości domyślne

```
def foo(a=5):
    return a + 1
```

```
foo(9)
```

```
10
```

```
foo()
```

```
6
```

☆ sort z limitem

Napisz funkcję sort, która sortuje elementy listy przekazanej jako argument i zwraca ją. Funkcja powinna posiadać dwie flagi (argumenty nazwane): reverse (domyślnie False) oraz limit (domyślnie False). Pierwsza z nich zmienia kierunek sortowania na przeciwny (czyli od największego elementu). Włączenie drugiej z nich powoduje, że zwracane jest tylko pierwsze pięć elementów.

```
sorted([1, 3, 2])
```

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

```
list(reversed([1, 2, 3]))
```

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

```
l = 'asdfqwer'
l[:5]
```

```
'asdfq'
```

Rozwiązanie

```
def sort(l, limit=False, reverse=False):
    l = sorted(l)
    if reverse:
        l = list(reversed(l))
    if limit:
        l = l[:5]
    return l
```

Oczekiwane zachowanie

```
sort('asdfghijk')
```

```
['a', 'd', 'f', 'g', 'h', 'i', 'j', 'k', 's']
```

```
sort('asdfghijk', limit=True)
```

```
sort('asdgfghijk', limit=True)
```

```
['a', 'd', 'f', 'g', 'h']
```

```
sort('asdgfghijk', reverse=True)
```

```
['s', 'k', 'j', 'i', 'h', 'g', 'f', 'd', 'a']
```

```
sort('asdgfghijk', limit=True, reverse=True)
```

```
['s', 'k', 'j', 'i', 'h']
```

```
sort([1, 3, 4, 7, 2, 4], reverse=True)
```

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

Parametry zmienne pozycyjne (*args)

```
def foo(args):  
    print(args)
```

```
foo(1, 2, 3)
```

```
-----  
TypeError          Traceback (most recent call last)  
<ipython-input-6-57a8fd452fb4> in <module>()  
      2     print(args)  
      3  
----> 4 foo(1, 2, 3)
```

```
TypeError: foo() takes exactly 1 argument (3 given)
```

```
def foo(*args):  
    print(args)
```

```
foo(1, 2, 3)
```

```
(1, 2, 3)
```

☆ sumall

Napisz funkcję, która przyjmuje dowolną liczbę argumentów (ale zawsze przynajmniej jeden) i zwraca ich sumę. Nie używaj funkcji sum.

Rozwiązanie

```
def sumall(*args):  
    if len(args) == 0:  
        raise ValueError("Pass at least one argument")  
    total = 0  
    for i in args:  
        total += i  
    return total
```

```
def sumall(*args):  
    assert len(args) == 0, "Pass at least one argument"  
    total = 0  
    for i in args:  
        total += i  
    return total
```

```
def sumall(first, *args):  
    for i in args:  
        first += i  
    return first
```

Oczekiwane zachowanie

```
sumall(5)
```

```
5
```

```
sumall(5, 6, 7)
```

```
18
```

```
sumall()
```

```
-----  
TypeError                      Traceback (most recent call last)  
<ipython-input-235-01c5f0c1ca42> in <module>()  
----> 1 sumall()  
  
TypeError: sumall() missing 1 required positional argument: 'first'
```

* przy wywołaniu

```
sumall(1, 2, 3)
```

```
6
```

```
I = [1, 2, 3]
```

```
sumall(5, 6, *I)
```

```
17
```

```
def bar(a, b, c):  
    return a + b + c
```

```
I = [3, 4, 5]  
bar(*I)
```

```
12
```

Parametry zmienne nazwane (**kwargs)

```
def foo(**kwargs):  
    print(kwargs)
```

```
foo(a=2, b=3)
```

```
{'a': 2, 'b': 3}
```

```
def bar(*args, c=5, **kwargs):  
    print(args)  
    print(c)  
    print(kwargs)
```

```
bar(2, 3, c=4, z=5)
```

```
(2, 3)
```

```
4
```

```
{'z': 5}
```

☆ dict_without_Nones

Napisz funkcję dict_without_Nones, która zachowuje się tak samo jak dict, ale usuwa klucze, dla których wartość jest None. Zadanie rozwiąż na dwa sposoby:

1. używając pętli for,
2. i używając dictionary comprehension: `{_: _ for _ in _ if _}`,

```
{'a': 2}
```

```
{'a': 2}
```

```
dict(a=2, b=None)
```

```
{'a': 2, 'b': None}
```

Rozwiązanie

```
def dict_without_Nones(**kwargs):  
    result = {}  
    for k, v in kwargs.items():  
        if v is not None:  
            result[k] = v  
    return result
```

```
def dict_without_Nones(**kwargs):
    return {k: v for k, v in kwargs.items()
            if v is not None}
```

Oczekiwane zachowanie

```
dict_without_Nones(a="1999", b="2000", c=None)
{'a': '1999', 'b': '2000'}
```

** przy wywołaniu

```
d = {'a': 1234, 'b': 2345, 'c': None}
```

```
dict_without_Nones(f=5, e=None, **d)
{'a': 1234, 'b': 2345, 'f': 5}
```

```
def bar(a, b, c):
    return a + b + c
```

```
d = {'a': 1, 'c': 5, 'b': 9}
bar(**d)
```

```
15
```

```
d = {'a': 2, 'b': 3}
e = {'b': 4, 'c': 5}
```

```
{**d, **e}
```

```
{'a': 2, 'b': 4, 'c': 5}
```

```
l = [1, 2, 3]
[2, 3, *l]
```

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

Wartości domyślne parametrów

☆ append_func

Napisz funkcję append_func, która dodaje element do listy. Element oraz lista powinny być jedynymi argumentami tej funkcji. Lista powinna być argumentem opcjonalnym. Jeżeli nie jest podana, należy utworzyć pustą listę. Funkcja powinna zwrócić listę z dodanym elementem.

Hint: użyj list.append(element).

Nieprawidłowe rozwiązanie

```
def append_func(item, seq=[]):
    print(id(seq))
    seq.append(item)
    return seq
```

```
append_func('e')
```

```
4405321672
```

```
['e']
```

```
append_func('f')
```

```
4405321672
```

```
['e', 'f']
```

Prawidłowe rozwiązanie

```
def append_func(item, seq=None):
    if seq is None:
```

```
seq = []
seq.append(item)
return seq
```

Oczekiwane zachowanie

```
append_func(3, [1, 2])
```

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

```
append_func('c', ['a', 'b'])
```

```
['a', 'b', 'c']
```

```
append_func('e')
```

```
['e']
```

```
append_func('f')
```

```
['f']
```

```
I = [1, 2]
```

```
append_func(3, I)
```

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

```
I
```

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

Funkcje anonimowe (lambda)

```
f = lambda x: x%2 == 0
```

```
def g(x):
    return x%2 == 0
```

```
f(3)
```

```
False
```

```
g(3)
```

```
False
```

filter i map

```
list(filter(f, [0, 1, 2, 3, 4, 5, 6]))
```

```
[0, 2, 4, 6]
```

```
filter(lambda x: x%2 == 0, range(10))
```

```
[0, 2, 4, 6, 8]
```

```
map(lambda x: x*2, range(10))
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

```
map(str, range(10))
```

```
['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
```

```
print(', '.join(map(lambda x: str(2*x), range(10))))
```

```
0, 2, 4, 6, 8, 10, 12, 14, 16, 18
```

```
print(', '.join( [str(2*elem) for elem in range(10)] ))
```

```
0, 2, 4, 6, 8, 10, 12, 14, 16, 18
```

☆ Suma liczb od 0 do 1000 podzielnych przez 3 lub 5

1. pętla for
2. list comprehension
3. filter i map

```
sum(filter(lambda x: x%3==0 or x%5==0, range(1001)))
```

234168

```
sum([x for x in range(1001) if x%3==0 or x%5==0])
```

234168

☆ Suma cyfr 2**1000

```
sum(map(int, str(2**1000)))
```

1366

```
sum([int(digit) for digit in str(2**1000)])
```

1366

★ Suma parzystych cyfr 2**1000

```
sum(filter(lambda x: x%2==0, map(int, str(2**1000))))
```

630

```
digits = map(int, str(2**1000))
sum(filter(lambda x: x%2==0, digits))
```

630

```
sum([int(digit) for digit in str(2**1000) if int(digit)%2==0])
```

630

```
sum([digit for digit
     in [int(digit) for digit in str(2**1000)]
     if digit%2==0])
```

630

☆ REPL

Zrefaktoryzuj poniższy kod poprzez podzielenie go na mniejsze funkcje oraz wykorzystanie słownika mapującego z nazw poleceń na funkcje wykonujące dane polecenie.

Kod początkowy

```
def repl():
    try:
        while True:
            line = input('>>> ')
            tokens = line.split(None, 1)

            try:
                command = tokens[0].lower()
            except IndexError:
                command = ""

            try:
                rest = tokens[1]
            except IndexError:
                rest = ""

            if command == 'eval':
                try:
                    result = eval(rest)
                except Exception as e:
                    print(type(e). __name__ )
```

```

        else:
            print(result)
        elif command == 'exit':
            raise KeyboardInterrupt
        elif command == "":
            pass
        else:
            print("Invalid command.")
    except KeyboardInterrupt:
        pass
    finally:
        print('Bye')

```

Rozwiążanie

```

def repl():
    try:
        while True:
            line = input('>>> ')
            command, rest = parse(line)
            execute(command, rest)
    except KeyboardInterrupt:
        pass
    finally:
        print('Bye')

def parse(line):
    tokens = line.split(None, 1)
    command = parse_command(tokens)
    rest = parse_rest(tokens)
    return command, rest

def parse_command(tokens):
    try:
        command = tokens[0]
    except IndexError:
        return ""
    else:
        return command.lower()

def parse_rest(tokens):
    try:
        return tokens[1]
    except IndexError:
        return ""

def execute(command, rest):
    try:
        func = commands[command]
    except KeyError:
        print(f"Nieznane polecenie: {command}")
    else:
        func(rest)

def do_eval(rest):
    try:
        result = eval(rest)
    except Exception as e:
        print(type(e).__name__)
    else:
        print(result)

def do_exit(rest):
    raise KeyboardInterrupt

def do_nothing(rest):
    pass

commands = {
    'eval': do_eval,
    'exit': do_exit,
    '': do_nothing,
}

```

Oczekiwane zachowanie

```
rep1@...  
>>> eval 2/0  
ZeroDivisionError  
>>> eval 2 + 2  
4  
>>> asdf  
Nieznane polecenie: asdf  
>>> exit  
Bye
```

Parsowanie Logów Apache

★ Suma bajtów

Chcemy sprawdzić, ile bajtów danych zostało przesłanych, sumując ostatnią kolumnę danych z pliku logów serwera Apache. W ostatniej kolumnie zamiast liczby może pojawić się -. Taki wiersz należy zignorować.

Plik logów Apache może wyglądać następująco:

```
81.107.39.38 - ... "GET /ply/ HTTP/1.1" 200 7587  
81.107.39.38 - ... "GET /favicon.ico HTTP/1.1" 404 133  
81.107.39.38 - ... "GET /ply/bookplug.gif HTTP/1.1" 200 23903  
81.107.39.38 - ... "GET /ply/ply.html HTTP/1.1" 200 97238  
81.107.39.38 - ... "GET /ply/example.html HTTP/1.1" 200 2359  
66.249.72.134 - ... "GET /index.html HTTP/1.1" 200 4447  
81.107.39.38 - ... "GET /ply/ HTTP/1.1" 304 -
```

W repozytorium znajduje się plik apache.log, który należy przetworzyć.

Pliki logów mogą ważyć bardzo dużo. W związku z tym wczytanie całego pliku do pamięci nie jest najlepszym pomysłem.

```
!cat apache.log
```

```
64.242.88.20 - - [07/Mar/2017:16:24:16 -0800] "GET / HTTP/1.0" 200 3395  
64.242.88.20 - - [07/Mar/2017:16:24:16 -0800] "GET /static/style.css HTTP/1.0" 200 3395  
64.242.88.20 - - [07/Mar/2017:16:24:16 -0800] "GET /static/logo.png HTTP/1.0" 200 293644  
64.242.88.20 - - [07/Mar/2017:16:24:30 -0800] "GET /login HTTP/1.0" 200 3395  
64.242.88.20 - - [07/Mar/2017:16:24:48 -0800] "POST /login?next=/dashboard/ HTTP/1.0" 302 -  
64.242.88.20 - - [07/Mar/2017:16:24:49 -0800] "GET /dashboard HTTP/1.0" 200 2002  
64.242.88.20 - - [07/Mar/2017:16:24:59 -0800] "GET /search?q=documentation HTTP/1.0" 200 2002  
64.242.88.20 - - [07/Mar/2017:16:25:13 -0800] "GET /doc/main HTTP/1.0" 200 2345  
64.242.88.20 - - [07/Mar/2017:16:25:23 -0800] "GET /doc/deployment HTTP/1.0" 200 28734  
64.242.88.20 - - [07/Mar/2017:16:26:56 -0800] "GET /doc/authentication HTTP/1.0" 200 22345  
65.123.12.28 - - [07/Mar/2017:16:27:32 -0800] "GET / HTTP/1.1" 200 3399  
64.242.88.20 - - [07/Mar/2017:16:28:05 -0800] "GET /doc/faq HTTP/1.0" 200 58462  
64.242.88.20 - - [07/Mar/2017:16:24:16 -0800] "GET /invalid-url HTTP/1.0" 404 7218
```

Rozwiązanie

```
import sys  
  
def main():  
    filename = sys.argv[1]  
    with open(filename, 'r') as s:  
        total = 0  
        for line in s:  
            line = line.strip()  
            bytes_as_str = line.split()[-1]  
            if bytes_as_str != '-':  
                total += int(bytes_as_str)  
        print(f"Total: {total}")  
  
if __name__ == "__main__":  
    main()
```

★ Podejście potokowe

Rozwiąż poprzednie zadanie w sposób potokowy, tzn. używając kilka wyrażeń listowych.

Hint

```
bytescolumn = ... # '7587', '133' lub '-' itd.
bytes = ... # 7587, 133 itd., `-` są pomijane
total = sum(bytes)
print("Total", total)
```

Rozwiązanie

```
with open("apache.log") as wwwlog:
    bytescolumn = (line.split()[-1] for line in wwwlog)
    bytes_ = (int(x) for x in bytescolumn if x != '-')
    sum_ = sum(bytes_)
    print(f"Total: {sum_}")
```

Total: 430336

☆★ apache2csv.py

Napisz skrypt apache2csv.py, który konwertuje logi Apache do pliku .csv. Dzięki temu będzie można wczytać logi do Excela.

Skrypt powinien przyjmować dwa argumenty: nazwę pliku wejściowego oraz wyjściowego. Jeżeli skrypt jest wywołany w złym sposobie, powinna wyświetlić się pomoc.

Kod początkowy

```
FIELD_NAMES = [
    'ip',
    'datetime',
    'method',
    # 'url',
    # 'protocol',
    'response_code',
    'content_length'
]
```

Rozwiązanie

```
%%writefile apache2csv.py
#!/usr/bin/env python
import sys

def display_help():
    print("Usage: apache2csv.py input_filename output_filename")

def split_line(line):
    parts = line.split()
    return {
        'ip': parts[0],
        'datetime': parts[3][1:] + ' ' + parts[4][:-1],
        'response_code': parts[8],
        'content_length': parts[9],
        'method': parts[5][1:],
        'url': parts[6],
        'protocol': parts[7][:-1],
    }

FIELD_NAMES = [
    'ip',
    'datetime',
    'method',
    'url',
    'protocol',
    'response_code',
    'content_length'
]

def run(input_filename, output_filename):
    if input_filename is None:
```

```

        input_stream = sys.stdin
    else:
        input_stream = open(input_filename, 'r')

    if output_filename is None:
        output_stream = sys.stdout
    else:
        output_stream = open(output_filename, 'w')

    with input_stream, output_stream:
        for line in input_stream:
            row = split_line(line)
            columns = [row[field] for field in FIELD_NAMES]
            csv_row = ','.join(columns)
            output_stream.write(csv_row + '\n')

def main(argv):
    try:
        input_filename = argv[1]
    except IndexError:
        input_filename = None

    try:
        output_filename = argv[2]
    except IndexError:
        output_filename = None

    if len(argv) > 3:
        display_help()
    else:
        run(input_filename, output_filename)

if __name__ == '__main__':
    main(sys.argv)

```

Overwriting apache2csv.py

Oczekiwane zachowanie

```

!chmod 700 apache2csv.py
!./apache2csv.py apache.log apache.csv
!cat apache.csv

64.242.88.20,07/Mar/2017:16:24:16 -0800,GET,/,HTTP/1.0,200,3395
64.242.88.20,07/Mar/2017:16:24:16 -0800,GET,/static/style.css,HTTP/1.0,200,3395
64.242.88.20,07/Mar/2017:16:24:16 -0800,GET,/static/logo.png,HTTP/1.0,200,293644
64.242.88.20,07/Mar/2017:16:24:30 -0800,GET,/login,HTTP/1.0,200,3395
64.242.88.20,07/Mar/2017:16:24:48 -0800,POST,/login?next=/dashboard/,HTTP/1.0,302,-
64.242.88.20,07/Mar/2017:16:24:49 -0800,GET,/dashboard,HTTP/1.0,200,2002
64.242.88.20,07/Mar/2017:16:24:59 -0800,GET,/search?q=documentation,HTTP/1.0,200,2002
64.242.88.20,07/Mar/2017:16:25:13 -0800,GET,/doc/main,HTTP/1.0,200,2345
64.242.88.20,07/Mar/2017:16:25:23 -0800,GET,/doc/deployment,HTTP/1.0,200,28734
64.242.88.20,07/Mar/2017:16:26:56 -0800,GET,/doc/authentication,HTTP/1.0,200,22345
65.123.12.28,07/Mar/2017:16:27:32 -0800,GET,/,HTTP/1.1,200,3399
64.242.88.20,07/Mar/2017:16:28:05 -0800,GET,/doc/faq,HTTP/1.0,200,58462
64.242.88.20,07/Mar/2017:16:24:16 -0800,GET,/invalid-url,HTTP/1.0,404,7218

```

```
!./apache2csv.py < apache.log
```

```

64.242.88.20,07/Mar/2017:16:24:16 -0800,GET,/,HTTP/1.0,200,3395
64.242.88.20,07/Mar/2017:16:24:16 -0800,GET,/static/style.css,HTTP/1.0,200,3395
64.242.88.20,07/Mar/2017:16:24:16 -0800,GET,/static/logo.png,HTTP/1.0,200,293644
64.242.88.20,07/Mar/2017:16:24:30 -0800,GET,/login,HTTP/1.0,200,3395
64.242.88.20,07/Mar/2017:16:24:48 -0800,POST,/login?next=/dashboard/,HTTP/1.0,302,-
64.242.88.20,07/Mar/2017:16:24:49 -0800,GET,/dashboard,HTTP/1.0,200,2002
64.242.88.20,07/Mar/2017:16:24:59 -0800,GET,/search?q=documentation,HTTP/1.0,200,2002
64.242.88.20,07/Mar/2017:16:25:13 -0800,GET,/doc/main,HTTP/1.0,200,2345
64.242.88.20,07/Mar/2017:16:25:23 -0800,GET,/doc/deployment,HTTP/1.0,200,28734
64.242.88.20,07/Mar/2017:16:26:56 -0800,GET,/doc/authentication,HTTP/1.0,200,22345
65.123.12.28,07/Mar/2017:16:27:32 -0800,GET,/,HTTP/1.1,200,3399
64.242.88.20,07/Mar/2017:16:28:05 -0800,GET,/doc/faq,HTTP/1.0,200,58462
64.242.88.20,07/Mar/2017:16:24:16 -0800,GET,/invalid-url,HTTP/1.0,404,7218

```

Jak zabrać się za to ćwiczenie? Podzielimy je na podzadania.

☆ split_line(line)

```
split_line('64.242.88.20 - - [07/Mar/2017:16:24:49 -0800] "GET /dashboard HTTP/1.0" 200 2002\n')
```

```
{'content_length': '2002',
'datetime': '07/Mar/2017:16:24:49 -0800',
'ip': '64.242.88.20',
'method': 'GET',
'protocol': 'HTTP/1.0',
'response_code': '200',
'url': '/dashboard'}
```

```
split_line('64.242.88.20 - - [07/Mar/2017:16:24:48 -0800] "POST /login?next=/dashboard/ HTTP/1.0" 302 -')
```

```
{'content_length': '-',
'datetime': '07/Mar/2017:16:24:48 -0800',
'ip': '64.242.88.20',
'method': 'POST',
'protocol': 'HTTP/1.0',
'response_code': '302',
'url': '/login?next=/dashboard'}
```

Hint

```
ip = '64.242.88.20'
method = 'POST'
d = {'ip': ip, 'method': method}
d
```

```
{'ip': '64.242.88.20', 'method': 'POST'}
```

```
d = {}
d['ip'] = '64.242.88.20'
d['method'] = 'POST'
d
```

```
{'ip': '64.242.88.20', 'method': 'POST'}
```

```
d = {
    'ip': '64.242.88.20',
    'method': 'POST',
}
d
```

```
{'ip': '64.242.88.20', 'method': 'POST'}
```

Rozwiązanie

```
def split_line(line):
    parts = line.split()
    return {
        'ip': parts[0],
        'datetime': parts[3][1:] + ' ' + parts[4][:-1],
        'response_code': parts[8],
        'content_length': parts[9],
        'method': parts[5][1:],
        'url': parts[6],
        'protocol': parts[7][:-1],
    }
```

☆ run(input_filename, output_filename)

- otwiera pliki wejściowy i wyjściowy
- iteruje po każdej linii z pliku wejściowego
- dla każdej linii wywołuje napisaną wcześniej funkcję split_line
- ze słownika zwróconego przez tą funkcję wybiera jedynie te kolumny, które określono w stałej FIELD_NAMES
- konkatenuje wszystkie dane i rozdziela je przecinkami (','.join(columns))
- dodaje tak utworzony nowy wiersz do pliku .csv

```
run('apache.log', 'apache.csv')
```

Hint

```
s = open('file.txt', 'w')
s.write('asdf\n')
s.write('qwer')
s.close()
```

```
!cat file.txt
```

```
asdf
qwer
```

```
log_record = split_line(line)
cols = [log_record[field_name] for field_name in FIELD_NAMES]
```

```
r = open('file.txt', 'w')
w = open('file.txt', 'r')
with r, w:
    pass
```

★ main(argv)

- dokonuje sprawdzenia, ile argumentów przekazano w wywołaniu programu
- wyświetla pomoc, jeżeli liczba argumentów się nie zgadza
- w przeciwnym razie wywołuje napisaną wcześniej funkcję run

```
main(['apache2csv.py'])
```

```
Usage: apache2csv.py input_filename output_filename
```

```
main(['apache2csv.py', 'invalid-number-of-arguments'])
```

```
Usage: apache2csv.py input_filename output_filename
```

```
main(['apache2csv.py', 'apache.log', 'apache.csv'])
```

★ argumenty opcjonalne

Załóż, że obie nazwy plików są opcjonalne. Jeżeli nie zostaną przekazane, powinniśmy czytać ze standardowego wejścia (sys.stdin) i pisać na standardowe wyjście (sys.stdout). Jeżeli podana jest nazwa tylko jednego pliku, należy założyć, że jest nazwa pliku wejściowego.

★ biblioteka csv

W standardowej bibliotece Pythona znajduje się moduł `csv`, który służy do czytania i tworzenia plików .csv. Ta biblioteka dba m.in. o poprawne escapowanie znaków specjalnych. Zauważ, że jeżeli gdziekolwiek w logach Apacha pojawi się przecinek, on nie jest escapowany w wyjściowym pliku .csv. W efekcie, zostanie on potraktowany jako rozpoczęcie nowej kolumny. Zastąp ręczne generowanie plików .csv tą biblioteką.

Programowanie Obiektowe

Podstawy

```
class BankAccount      # old-style class in Python 2
class BankAccount(object) # new-style class in Python 2
class BankAccount # implicit object # new-style class in Python 3
class BankAccount(object)      # new-style class in Python 3

class BankAccount(object): # Python 2 i 3
    def __init__(self):
        print('__init__')
        self.balance = 0

    def deposit(self, amount):
        self.balance += amount
```

```
def withdraw(self, amount):
    self.balance -= amount
```

```
def bonus(self):
    self.deposit(1000)
```

```
account = BankAccount() # BankAccount.__init__
```

```
__init__
```

```
account
```

```
<__main__.BankAccount at 0x1065bc160>
```

```
account.deposit(1000)
```

```
BankAccount.deposit(account, 1000)
```

```
account.balance
```

```
2000
```

Instancje są niezależne od siebie:

```
another_account = BankAccount()
```

```
__init__
```

```
another_account.balance
```

```
0
```

```
another_account.withdraw(500)
```

```
another_account.balance
```

```
-500
```

```
account.balance
```

```
1000
```

```
account.balance = 10
```

```
account.balance
```

```
10
```

Atrybuty klas i instancji

Atrybuty klasy są wspólne dla wszystkich instancji. Czasami używa się ich jako domyślnych wartości dla atrybutów instancji.

```
class Person(object):
    first_name = 'John'
    last_name = 'Smith'
    phones = []

# def __init__(self):
#     # Pola/atributy instancji
#     self.first_name = 'John'
#     self.phones = []

def foo(self):
    pass

p1 = Person()
p2 = Person()
```

```
p1.first_name = 'Bob'
p1.first_name, p2.first_name
```

```
('Bob', 'John')
```

```
Person.last_name = 'Williams'
n1 last name n2 last name
```

```
p1.first_name, p2.first_name
```

```
('Williams', 'Williams')
```

Najlepiej jest unikać używania atrybutów klas do implementacji domyślnych wartości dla atrybutów instancji, ponieważ wiąże się to z pułapką, jeżeli domyślona wartość jest mutowalna.

```
p1.phones.append(+48123456789)
```

```
p1.phones, p2.phones
```

```
[ [+48123456789], [+48123456789] ]
```

```
Person.__dict__
```

```
mappingproxy({ '__dict__': <attribute '__dict__' of 'Person' objects>,
    '__doc__': None,
    '__module__': '__main__',
    '__weakref__': <attribute '__weakref__' of 'Person' objects>,
    'first_name': 'John',
    'foo': <function __main__.Person.foo>,
    'last_name': 'Williams',
    'phones': [ '+48123456789' ]})
```

```
p2.__dict__
```

```
{}
```

```
p1.__dict__
```

```
{'first_name': 'Bob'}
```

```
p1.first_name, p2.first_name
```

```
('Bob', 'John')
```

```
del p1.first_name
```

```
p1.first_name
```

```
'John'
```

@property

Dekorator @property umożliwia kontrolę dostępu do atrybutu. Zastępuje gettery i settery.

```
class BankAccount(object):
    def __init__(self, initial_balance):
        self._balance = initial_balance

    @property
    def balance(self):
        print('get')
        return self._balance

    @balance.setter
    def balance(self, value):
        if value < 0:
            raise ValueError('Invalid, negative balance')
        self._balance = value

    # rowniez:
    ...

    def asdf(self, value):
        if value < 0:
            raise ValueError('Invalid, negative balance')
        self._balance = value

    balance = balance.setter(asdf)
    ...
```

```
account = BankAccount(100.0)
```

```
account.balance
```

```
get
```

```
100.0
```

```
account.balance = 10
```

```
account.balance = -10
```

```
-----  
ValueError      Traceback (most recent call last)  
<ipython-input-35-d0154f3dd9af> in <module>()  
----> 1 account.balance = -10
```

```
<ipython-input-29-43a3305e33e1> in balance(self, value)  
  11     def balance(self, value):  
  12         if value < 0:  
--> 13             raise ValueError('Invalid, negative balance')  
  14         self._balance = value  
  15
```

```
ValueError: Invalid, negative balance
```

```
account.balance
```

```
get
```

```
10
```

☆ RecentlyUsedList

Zaimplementuj klasę RecentlyUsedList - strukturę danych, która przechowuje elementy w ustalonej kolejności, podobnie jak lista. W przeciwnieństwie do listy:

- nowe elementy dodawane są na początek, a nie na koniec
- jeżeli dany element jest dodany ponownie, to jest on przesuwany na początek; w efekcie, RecentlyUsedList nigdy nie posiada duplikatów.

Rozwiązanie

```
class RecentlyUsedList(object):  
    def __init__(self, initial_list=None):  
        self._list = []  
        if initial_list is not None:  
            for item in initial_list:  
                self.insert(item)  
  
    @property  
    def length(self):  
        return len(self._list)  
  
    def insert(self, item):  
        # if item in self._list:  
        #     self._list.remove(item)  
        try:  
            self._list.remove(item)  
        except ValueError:  
            pass  
        self._list.insert(0, item)  
  
    def get(self, index):  
        return self._list[index]  
  
    def to_string(self):  
        return f'RecentlyUsedList({self._list})'
```

Oczekiwane zachowanie

```
rul = RecentlyUsedList()
```

```
rul.length
```

```
0
```

```
rul.length = 3
```

```
-----  
AttributeError      Traceback (most recent call last)
```

```
<ipython-input-147-830cc842939a> in <module>()
----> 1 rul.length = 3
```

AttributeError: can't set attribute

```
rul.insert('first')
rul.insert('second')
```

```
rul.get(0)
```

'second'

```
rul.get(1)
```

'first'

```
rul.get(2)
```

```
IndexError                                Traceback (most recent call last)
<ipython-input-48-1ef5b5ed34c6> in <module>()
----> 1 rul.get(2)
```

```
<ipython-input-42-bb52832d8d4b> in get(self, index)
 15
 16     def get(self, index):
--> 17         return self._list[index]
 18
 19     def to_string(self):
```

IndexError: list index out of range

```
rul.to_string()
```

"RecentlyUsedList(['second', 'first'])"

```
rul.insert('third')
rul.to_string()
```

"RecentlyUsedList(['third', 'second', 'first'])"

```
rul.insert('second')
rul.to_string()
```

"RecentlyUsedList(['second', 'third', 'first'])"

```
rul = RecentlyUsedList(['a', 'b', 'c', 'a'])
rul.to_string()
```

"RecentlyUsedList(['a', 'c', 'b'])"

☆ BankAccount

- Zaimplementuj klasę BankAccount reprezentującą konto bankowe.
- Konto bankowe ma identyfikator (id).
- Na koncie może znajdować się dowolna liczba środków lub może mieć debet (property balance) .
- Konto może znajdować się w jednym z dwóch stanów (property state): "normal" lub "overdraft".
- Na koncie można zdeponować środki (metoda deposit).
- Z konta można pobrać środki (metoda withdraw) jeśli tylko jest w stanie "normal", nawet jeśli nie ma na nim wystarczająco dużo środków. Jeżeli konto jest w stanie "overdraft", wówczas rzucony jest wyjątek ValueError zawierający informację o identyfikatorze konta.
- Na koncie naliczają się odsetki (metoda pay_interest). 10% od zgromadzonych środków. Jeśli konto jest w stanie "overdraft", wówczas naliczane są 20% odsetki.

Rozwiążanie

```
class BankAccount(object):
    INTEREST_RATE = {
        'Normal': 0.1,
        'Overdraft': 0.2,
    }

    def __init__(self, id_):
        self.id = id_
        self._balance = 0
```

```
def deposit(self, amount):
    self._balance += amount

@property
def state(self):
#     if self._balance >= 0:
#         return 'Normal'
#     else:
#         return 'Overdraft'
#     (_ if _ else _)
    return 'Normal' if self._balance >= 0 else 'Overdraft'

@property
def balance(self):
    return self._balance

def withdraw(self, amount):
    if self.state == 'Normal':
        self._balance -= amount
    else:
        msg = f'Insufficient funds for bank account: {self.id}'
        raise ValueError(msg)

def pay_interest(self):
    self._balance += self._balance * self.INTEREST_RATE[self.state]
```

Oczekiwane zachowanie

```
account = BankAccount(1234)
```

Konto bankowe ma identyfikator (id).

```
account.id
```

1234

Początkowy stan konta to 0.

```
account.balance
```

0

Konto może być w stanie 'Normal' lub 'Overdraft', jeżeli jest na nim debet.

```
account.state
```

'Normal'

Oczekiwane zachowanie - deposit()

```
account.deposit(1000)
```

```
account.balance
```

1000

```
account.state
```

'Normal'

Oczekiwane zachowanie - pay_interest() przy dostępnych środkach na koncie

```
account.balance
```

1000

```
account.pay_interest()
```

```
account.balance
```

1100.0

Oczekiwane zachowanie - withdraw() przy dostępnych środkach na koncie

```
account.withdraw(1600)
```

```
account.balance
```

```
-500.0
```

```
account.state
```

```
'Overdraft'
```

Oczekiwane zachowanie - withdraw() przy debecie

```
account.withdraw(100)
```

```
-----  
ValueError                                Traceback (most recent call last)  
<ipython-input-125-bf10e29dd2a1> in <module>()  
----> 1 account.withdraw(100)
```

```
<ipython-input-101-db8ad91844dc> in withdraw(self, amount)  
    21     else:  
    22         msg = 'Insufficient funds for bank account: {}'.format(self.id)  
--> 23         raise ValueError(msg)  
    24  
    25     def pay_interest(self):
```

```
ValueError: Insufficient funds for bank account: 1234
```

Oczekiwane zachowanie - pay_interest() przy debecie

```
account.balance
```

```
-500.0
```

```
account.pay_interest()
```

```
account.balance
```

```
-600.0
```

isinstance()

Do sprawdzenia, czy dany obiekt jest danej klasy, służy wbudowana funkcja isinstance.

```
isinstance(2, int)
```

```
True
```

```
isinstance(2, float)
```

```
False
```

```
isinstance(account, BankAccount)
```

```
True
```

Drugi argument może być listą klas - wtedy pierwszy argument musi być instancją którejś z tej klas.

```
isinstance(3, (int, float))
```

```
True
```

```
isinstance(3.0, (int, float))
```

```
True
```

```
isinstance('bla', (int, float))
```

```
False
```

type()

Wbudowana funkcja type() zwraca klasę dla danego obiektu.

```
type(2)
<type 'int'>

type(account)
<class '__main__.BankAccount'>

isinstance(2, type(2))
True

isinstance(account, BankAccount)
True

isinstance(2, object)
True
```

☆ Person

Zaimplementuj klasę Person reprezentującą pojedynczą osobę. Klasa powinna mieć dwa pola: first_name i last_name, przy czym ostatnie pole jest opcjonalne (tzn. może mieć wartość None). Klasa powinna udostępniać możliwość pobrania pełnego imienia i nazwiska jako jednego napisu (Person.full_name). Klasa powinna walidować w __init__, czy pola mają poprawne wartości - tzn. są napisami i każdy z nich ma minimum dwa znaki.

Rozwiązanie

```
class Person(object):
    def __init__(self, first_name, last_name=None):
        if not isinstance(first_name, str):
            raise TypeError('First name must be a string or unicode.')
        if len(first_name) <= 1:
            raise ValueError('First name must have at least two characters.')

        if last_name is not None:
            if not isinstance(last_name, str):
                raise TypeError('Last name must be a string or unicode or None.')
            if len(last_name) <= 1:
                raise ValueError('Last name must have at least two characters.')

        self.first_name = first_name
        self.last_name = last_name

    @property
    def full_name(self):
        if self.last_name is None:
            return self.first_name
        else:
            return '{}, {}'.format(self.first_name, self.last_name)
```

Oczekiwane zachowanie

```
p = Person('Jan', 'Kowalski')
p.first_name, p.last_name
```

```
('Jan', 'Kowalski')
```

```
p.full_name
```

```
'Jan Kowalski'
```

```
p2 = Person('Jan')
p2.first_name, p2.last_name
```

```
('Jan', None)
```

```
p2.full_name
```

```
'Jan'
```

Oczekiwane zachowanie - obsługa błędów

```
Person(23, 'Kowalski')
```

Traceback (most recent call last):

```
File "<ipython-input-137-611f50ae3b59>", line 1, in <module>
  Person(23, 'Kowalski')
```

```
File "<ipython-input-132-97a603ec03c5>", line 4, in __init__
  raise TypeError('First name must be a string or unicode.')
```

`TypeError: First name must be a string or unicode.`

```
Person('J', 'Kowalski')
```

Traceback (most recent call last):

```
File "<ipython-input-138-250388e0573e>", line 1, in <module>
  Person('J', 'Kowalski')
```

```
File "<ipython-input-132-97a603ec03c5>", line 6, in __init__
  raise ValueError('First name must have at least two characters.')
```

`ValueError: First name must have at least two characters.`

```
Person('Jan', 23)
```

Traceback (most recent call last):

```
File "<ipython-input-139-4ba85fffa19e>", line 1, in <module>
  Person('Jan', 23)
```

```
File "<ipython-input-132-97a603ec03c5>", line 8, in __init__
  raise TypeError('Last name must be a string or unicode or None.')
```

`TypeError: Last name must be a string or unicode or None.`

```
Person('Jan', 'K')
```

Traceback (most recent call last):

```
File "<ipython-input-140-42ad5d522c76>", line 1, in <module>
  Person('Jan', 'K')
```

```
File "<ipython-input-132-97a603ec03c5>", line 10, in __init__
  raise ValueError('Last name must have at least two characters.')
```

`ValueError: Last name must have at least two characters.`

Dziedziczenie pojedyncze

```
class Base(object):
    def base_method(self):
        print('base method')

class Derived(Base):
    def derived_method(self):
        print('derived method')
```

```
issubclass(Derived, Base)
```

True

```
base_instance = Base()
derived_instance = Derived()
```

```
isinstance(derived_instance, Base)
```

True

```
base_instance.base_method()
```

base method

```
derived_instance.derived_method()
```

```
derived method
```

Klasa pochodna dziedziczy metody z klasy bazowej. Oznacza ta, że metody zdefiniowane w klasie bazowej są dostępne także w klasie pochodnej:

```
derived_instance.base_method()
```

```
base method
```

Wbudowana funkcja `dir(obj)` określa wszystkie atrybuty (zarówno metody jak i stan obiektów) dostępne w obiekcie `obj`.

```
dir(derived_instance)
```

```
['__class__',
 '__delattr__',
 '__dict__',
 '__doc__',
 '__format__',
 '__getattribute__',
 '__hash__',
 '__init__',
 '__module__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 'base_method',
 'derived_method']
```

super()

`super()` używamy, jeżeli przeciążamy jakąś metodę, a w środku niej chcemy wywołać implementację z klasy bazowej.

```
class Base(object):
    def base_method(self):
        print('base method')
        return 42

class Derived(Base):
    def derived_method(self):
        print('derived method')

    def base_method(self):
        super(Derived, self).base_method() # Python 2
        # super().base_method() # Python 3
        print("base method customized")
```

```
derived_instance = Derived()
derived_instance.base_method()
```

```
<super: <class 'Derived'>, <Derived object>>
base method
42
base method customized
```

`super()` najczęściej używa się w `__init__`.

```
class Point(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y

class Point3D(Point):
    def __init__(self, x, y, z):
        super(Point3D, self).__init__(x, y)
        self.z = z
```

```
n = Point(3, 0, 4, 0)
```

```
p = Point(3.0, 4.0)
```

```
p.x, p.y
```

```
(3.0, 4.0)
```

```
p3d = Point3D(3.0, 4.0, 5.0)
```

```
p3d.x, p3d.y, p3d.z
```

```
(3.0, 4.0, 5.0)
```

Klasy wyjątków

Python posiada wiele wbudowanych klas wyjątków, z których najpopularniejsze to ValueError, TypeError, KeyError i IndexError. Można stworzyć nową klasę wyjątków poprzez dziedziczenie po klasie bazowej Exception.

```
class MyException(Exception):  
    pass
```

```
raise MyException()
```

```
MyException                                Traceback (most recent call last)  
<ipython-input-80-c80dc8b33025> in <module>()  
----> 1 raise MyException()
```

```
MyException:
```

```
raise MyException('info')
```

```
MyException                                Traceback (most recent call last)  
<ipython-input-81-38a92a7c0a60> in <module>()  
----> 1 raise MyException('info')
```

```
MyException: info
```

☆ CustomizedException

Napisz klasę CustomizedException, dla której litery komunikatu są zamienione na wielkie.

Rozwiązanie

```
class CustomizedException(Exception):  
    def __init__(self, message):  
        msg = message.upper()  
        super(CustomizedException, self).__init__(msg)
```

Oczekiwane zachowanie

```
raise CustomizedException('info')
```

```
CustomizedException      Traceback (most recent call last)  
<ipython-input-170-b70fa668776c> in <module>()  
----> 1 raise CustomizedException('info')
```

```
CustomizedException: INFO
```

☆ Baza osób

Zaimplementuj następujące klasy.

klasa	klasa bazowa	pola	metody
Person	object	name, surname, number	-
Student	Person	name, surname, number, student_type, classes	enrol(course)
StaffMember	Person	name, surname, number, employment_type	-
Lecturer	StaffMember	name, surname, number, employment_type, courses_taught	assign_teaching(course)

- student type powinno być napisem 'undergraduate' lub 'postgraduate'.

- employment_type powinno być napisem 'permanent' lub 'temporary'
- classes i courses_taught powinny być listami kursów reprezentowanych przez napisy
- zaimplementuj validację dla enumów (student_type i employment_type)

Rozwiązanie

```
STUDENT_TYPES = ('undergraduate', 'postgraduate')
EMPLOYMENT_TYPES = ('permanent', 'temporary')
```

```
class Person(object):
    def __init__(self, name, surname, number):
        self.name = name
        self.surname = surname
        self.number = number

class Student(Person):
    def __init__(self, student_type, *args, **kwargs):
        super(Student, self).__init__(*args, **kwargs)
        if student_type not in STUDENT_TYPES:
            raise ValueError('student_type must be one of {}'.format(STUDENT_TYPES))
        self.student_type = student_type
        self.classes = []

    def enrol(self, course):
        self.classes.append(course)

class StaffMember(Person):
    def __init__(self, employment_type, *args, **kwargs):
        super(StaffMember, self).__init__(*args, **kwargs)
        if employment_type not in EMPLOYMENT_TYPES:
            raise ValueError('employment_type must be one of {}'.format(EMPLOYMENT_TYPES))
        self.employment_type = employment_type

class Lecturer(StaffMember):
    def __init__(self, *args, **kwargs):
        super(Lecturer, self).__init__(*args, **kwargs)
        self.courses_taught = []

    def assign_teaching(self, course):
        self.courses_taught.append(course)
```

Oczekiwane zachowanie - Person

```
person = Person('Jane', 'Smith', '123456')
person.name, person.surname, person.number

('Jane', 'Smith', '123456')
```

Oczekiwane zachowanie - Student

```
student = Student('postgraduate', 'Jane', 'Smith')
student.name, student.surname, student.number, student.student_type, student.classes

('Jane', 'Smith')
{}

-----
TypeError           Traceback (most recent call last)
<ipython-input-175-3bdf02af8353> in <module>()
----> 1 student = Student('postgraduate', 'Jane', 'Smith')
      2 student.name, student.surname, student.number, student.student_type, student.classes

<ipython-input-173-f371047fdc6f> in __init__(self, student_type, *args, **kwargs)
    14     print(args)
    15     print(kwargs)
--> 16     super(Student, self).__init__(*args, **kwargs)
    17     if student_type not in STUDENT_TYPES:
    18         raise ValueError('student_type must be one of {}'.format(STUDENT_TYPES))

TypeError: __init__() takes exactly 4 arguments (3 given)
```

```
student.enrol('Math 101')
```

```
student.classes
```

```
['Math 101']
```

Oczekiwane zachowanie - StaffMember

```
staff_member = StaffMember('permanent', 'Jane', 'Smith', 'SMTJNX045')
```

```
staff_member.name, staff_member.surname, staff_member.number, staff_member.employment_type
```

```
('Jane', 'Smith', 'SMTJNX045', 'permanent')
```

Oczekiwane zachowanie - Lecturer

```
lecturer = Lecturer('permanent', 'Bob', 'Jones', '123456789')
```

```
lecturer.name, lecturer.surname, lecturer.number, lecturer.employment_type, lecturer.courses_taught
```

```
('Bob', 'Jones', '123456789', 'permanent', [])
```

```
lecturer.assign_teaching('History 101')
```

```
lecturer.courses_taught
```

```
['History 101']
```

Oczekiwane zachowanie - walidacja

```
student = Student('blah', 'Jane', 'Smith', 'SMTJNX045')
```

```
-----  
ValueError Traceback (most recent call last)
```

```
<ipython-input-2-62c3b3177ce2> in <module>()
```

```
---> 1 student = Student('blah', 'Jane', 'Smith', 'SMTJNX045')
```

```
<ipython-input-1-e45924ccacc2> in __init__(self, student_type, *args, **kwargs)
```

```
14     super(Student, self).__init__(*args, **kwargs)
```

```
15     if student_type not in STUDENT_TYPES:
```

```
--> 16         raise ValueError('student_type must be one of {}'.format(STUDENT_TYPES))
```

```
17     self.student_type = student_type
```

```
18     self.classes = []
```

```
ValueError: student_type must be one of ('undergraduate', 'postgraduate')
```

```
lecturer = Lecturer('blah', 'Bob', 'Jones', '123456789')
```

```
-----  
ValueError Traceback (most recent call last)
```

```
<ipython-input-3-39f261a0ee21> in <module>()
```

```
---> 1 lecturer = Lecturer('blah', 'Bob', 'Jones', '123456789')
```

```
<ipython-input-1-e45924ccacc2> in __init__(self, *args, **kwargs)
```

```
32 class Lecturer(StaffMember):
```

```
33     def __init__(self, *args, **kwargs):
```

```
--> 34         super(Lecturer, self).__init__(*args, **kwargs)
```

```
35         self.courses_taught = []
```

```
36
```

```
<ipython-input-1-e45924ccacc2> in __init__(self, employment_type, *args, **kwargs)
```

```
26     super(StaffMember, self).__init__(*args, **kwargs)
```

```
27     if employment_type not in EMPLOYMENT_TYPES:
```

```
--> 28         raise ValueError('employment_type must be one of {}'.format(EMPLOYMENT_TYPES))
```

```
29     self.employment_type = employment_type
```

```
30
```

```
ValueError: employment_type must be one of ('permanent', 'temporary')
```

Metody Specjalne

☆ RecentlyUsedList 2

Przepisz istniejącą implementację klasy RecentlyUsedList tak, aby implementowała tym razem inny interfejs, składający się głównie z metod specjalnych. Użyteczna będzie tutaj [dokumentacja dotycząca listy metod specjalnych](#).

Rozwiązanie

```
class RecentlyUsedList(object):
    def __init__(self):
        self._list = []

    def __len__(self):
        return len(self._list)

    def insert(self, item):
        try:
            self._list.remove(item)
        except ValueError:
            pass
        self._list.insert(0, item)

    def __getitem__(self, index):
        return self._list[index]

    def __repr__(self):
        return 'RecentlyUsedList({})'.format(self._list)
```

Oczekiwane zachowanie

```
rul = RecentlyUsedList()
```

```
len(rul) # rul.__len__()
```

```
0
```

```
rul
```

```
RecentlyUsedList([])
```

```
rul.insert('first')
rul.insert('second')
```

```
rul[0] # rul.__getitem__(0)
```

```
'second'
```

```
rul[1]
```

```
'first'
```

```
rul[2]
```

```
-----  
IndexError      Traceback (most recent call last)  
<ipython-input-71-9af6dc8555ff> in <module>()  
----> 1 rul[2]
```

```
<ipython-input-64-3ea8935eaf40> in __getitem__(self, index)
 14
 15     def __getitem__(self, index):
--> 16         return self._list[index]
 17
 18     def __repr__(self):
```

```
IndexError: list index out of range
```

```
print(repr(rul)) # rul.__repr__
```

```
RecentlyUsedList(['second', 'first'])
```

```
print(str(rul)) # rul.__str__
```

```
RecentlyUsedList(['second', 'first'])
```

```
print(str('asdf'))
```

```
asdf
```

```
rul.insert('third')
rul
```

```
RecentlyUsedList(['third', 'second', 'first'])
```

```
rul.insert('second')
rul
RecentlyUsedList(['second', 'third', 'first'])
```

☆ Record

1. Napisz klasę zachowującą się jak słownik, ale wykorzystującą składnię dict[key] zamiast dict[key].
2. Użyj kompozycji, tzn. instancje Record powinny mieć pole dict.
3. W środku metod specjalnych dodaj printy

Podpowiedź

```
class Record(object):
    def __init__(self): pass
    def __getattr__(self, name): pass # self.name
    def __setattr__(self, name, value): pass # self.name = value

# Druga część ćwiczenia:
def __getitem__(self, name): pass # self[name]
def __setitem__(self, name, value): pass
```

Rozwiązanie

```
class Record(object):
    def __init__(self):
        # Nie możemy użyć poniższego kodu:
        #   self._d = {}
        # Ponieważ zakończyłby się on rekurencyjnym wywoływaniem metody __setattr__
        super(Record, self).__setattr__('_dict', {})

    def __getattr__(self, name):
        print('get', name)
        self._dict
        return self._dict[name]

    def __setattr__(self, name, value):
        print('set', name, 'to', value)
        self._dict
        self._dict[name] = value
```

Oczekiwane zachowanie

```
person = Record()

person.first_name = 'John' # person.__setattr__('first_name', 'John')

set first_name to John

print(person.first_name) # person.__getattr__('first_name')

get first_name
John

person['first_name'] = 'John'
```

☆ Klasa jako Funkcja

Funkcja factorial liczy rekurencyjnie silnię. factorial(5) == 5*4*3*2*1 == 120.

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

```
factorial(5)
```

Zamień tę funkcję na klasę.

Podpowiedź

```
class Foo(object):
    def __call__(self, arg):
        print("AAA", arg)
```

```
foo = Foo()
foo(42) # foo.__call__(42)
```

AAA 42

Kod początkowy

```
class Factorial(object):
    def __init__(...): pass
    def __call__(...): pass
```

Rozwiązanie

```
class Factorial(object):
    def __call__(self, n):
        if n == 0:
            return 1
        else:
            return n * self(n-1)
```

Oczekiwane zachowanie

```
Factorial()(6)
```

720

```
inst = Factorial()
inst(6)
```

720

★ Klasa jako Funkcja 2

Funkcja cached_factorial liczy rekurencyjnie silnię, ale cachuje wyniki.

```
cache = {0: 1}
def cached_factorial(n):
    if n not in cache:
        cache[n] = n * cached_factorial(n-1)
    return cache[n]
```

```
cached_factorial(5)
```

120

```
cache
```

{0: 1, 1: 1, 2: 2, 3: 6, 4: 24, 5: 120}

Zamień tę funkcję na klasę CachedFactorial. Nie używaj zmiennej globalnej cache. Zamiast tego, przechowuj wyniki w instancji klasy CachedFactorial.

Rozwiązanie

Oczekiwane zachowanie

```
cf = CachedFactorial()
cf(5)
```

120

```
cf._cache
```

```
{1: 1, 2: 2, 3: 6, 4: 24, 5: 120}
```

@classmethod

☆ Date

Napisz klasę Date, którą można utworzyć na dwa sposoby:

1. wywołując "konstruktor" `__init__` i przekazując trzy liczby (dzień, miesiąc i rok)
2. wywołując metodę klasy `from_string` i przekazując napis reprezentujący datę w formacie DD-MM-YYYY.

Hint: do sparsowania daty użyj `str.split('-')`.

Hint

```
date = '20-01-2016'
date.split('-')
```

```
['20', '01', '2016']
```

Kod początkowy

```
class Date(object):
    def __init__(...):
        ...

    @classmethod
    def from_string(cls, ...):
        ...
```

Rozwiązanie

Oczekiwane zachowanie

```
d1 = Date(20, 1, 2016)
d1.day, d1.month, d1.year
```

```
(20, 1, 2016)
```

```
d2 = Date.from_string('20-01-2016')
d2.day, d2.month, d2.year
```

```
(20, 1, 2016)
```

★ ExtendedDate

Napisz klasę ExtendedDate dziedziczącą po Date. Obiekty klasy ExtendedDate powinny być porównywalne.

Rozwiązanie

Oczekiwane zachowanie

```
dt = ExtendedDate.from_string('20-01-2016')
dt
<__main__.ExtendedDate at 0x7f6056ea6b90>
```

```
dt2 = ExtendedDate.from_string('20-01-2016')
dt == dt2
```

True

```
dt3 = ExtendedDate.from_string('10-01-2016')
dt == dt3
```

False

@staticmethod

Metoda statyczna to po prostu funkcja w przestrzeni nazw klasy. Taką funkcję należy udekorować @staticmethod. Taka funkcja nie przyjmuje instancji klasy self ani samej klasy cls.

```
from math import sqrt

class Point(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y

    @staticmethod
    def get_distance(p1, p2):
        return sqrt((p1.x-p2.x)**2 + (p1.y-p2.y)**2)
```

```
p1 = Point(2, 2)
p2 = Point(5, 6)
Point.get_distance(p1, p2)
```

5.0

Dziedziczenie wielokrotne

```
class A(object):
    def __init__(self):
        print("A")

class B(A):
    def __init__(self):
        super(B, self).__init__()
        print("B")

class C(A):
    def __init__(self):
        super(C, self).__init__()
        print("C")

class D(B, C):
    def __init__(self):
        super(D, self).__init__()
        print("D")
```

D()

A
C
B
D

<__main__.D at 0x7ff0b429cc0>

Method Resolution Order

```
D.__mro__
(__main__.D, __main__.B, __main__.C, __main__.A, object)
```

Czasami nie da się utworzyć sensownego MRO:

```
class A(object): pass
class B(object): pass
class C(A, B): pass
```

```
class D(B, A): pass  
class E(C, D): pass
```

```
-----  
TypeError Traceback (most recent call last)  
<ipython-input-21-53cb8f507db8> in <module>()  
      3 class C(A, B): pass  
      4 class D(B, A): pass  
----> 5 class E(C, D): pass
```

TypeError: Error when calling the metaclass bases
 Cannot create a consistent method resolution
order (MRO) for bases B, A

Mixins (klasy domieszkowe)

Klasy domieszkowe (mixins) to klasy, które dostarczają określoną funkcjonalność innym klasom (poprzez mechanizm wielokrotnego dziedziczenia). Nie są samodzielonymi klasami i, w związku z tym, nie tworzy się instancji klas domieszkowych. Zazwyczaj nazwa takiej klasy kończy się sufiksem Mixin (np. ComparableMixin), ale nie jest to bezwględnie obowiązująca konwencja.

Definiując klasę i wymieniając jej rodziców, należy pamiętać, aby najpierw wymienić wszystkie klasy domieszkowe, a dopiero na końcu podać klasę bazową (chyba że jest nią object).

☆ ComparableMixin

W przypadku porównywania obiektów, wywoływana jest jedna z sześciu specjalnych metod (`__lt__`, `__le__`, `__eq__`, `__ne__`, `__gt__` lub `__ge__`). Wystarczy jednak zdefiniować dwie z nich (np. `__le__` i `__eq__`), a pozostałe porównania to odpowiednia kombinacja tych dwóch metod:

```
a != b <=> not (a == b)  
a < b <=> (a <= b) and not (a == b)  
a > b <=> not (a <= b)  
a >= b <=> (a == b) or not (a <= b)
```

Zaimplementuj i przetestuj klasę domieszkową ComparableMixin. Jej użycie powoduje, że wystarczy zdefiniować metody `__le__` i `__eq__`, aby obiekty klasy dziedziczącej po ComparableMixin mogły być porównywane.

Kod początkowy

```
class MyInteger(ComparableMixin):  
    def __init__(self, i):  
        self.i = i  
    def __le__(self, other):  
        return self.i <= other.i  
    def __eq__(self, other):  
        return self.i == other.i  
  
class ComparableMixin(object):  
    pass
```

Rozwiązanie

Oczekiwane zachowanie

```
MyInteger(1) > MyInteger(0)
```

```
True
```

★ Tutor

Wykorzystaj napisaną wcześniej implementację klasy Person:

```
class Person(object):  
    def __init__(self, name, surname, number):  
        self.name = name  
        self.surname = surname  
        self.number = number
```

Następnie zaimplementuj następujące mixiny:

nazwa	pola	metody
LearnerMixin	classes	enrol(course)
TeacherMixin	courses_taught	assign_teaching(course)

Używając tych mixinów, zaimplementuj klasę Tutor, która ma zarówno metodę enrol, jak i assign_teaching.

Rozwiązanie

Oczekiwane zachowanie

```
jane = Tutor("Jane", "Smith", "SMTJNX045")
jane.name, jane.classes, jane.courses_taught
```

```
('Jane', [], [])
```

```
jane.enrol('Math 101')
jane.name, jane.classes, jane.courses_taught
```

```
('Jane', ['Math 101'], [])
```

```
jane.assign_teaching('History 101')
jane.name, jane.classes, jane.courses_taught
```

```
('Jane', ['Math 101'], ['History 101'])
```

Organizacja Kodu

Moduły

Modułem jest plik źródłowy z rozszerzeniem .py zawierający kod Pythona. Moduł może zawierać definicje funkcji, klas, a także niezależny od nich kod. Moduł może zawierać dokumentację informującą o sposobie jego działania i zastosowaniach.

Wewnątrz modułu jego nazwa (nazwa pliku bez rozszerzenia .py) jest dostępna pod globalną zmienną `__name__`, pod warunkiem że moduł jest importowany. W przeciwnym razie jego nazwa to "`__main__`".

Pakiety

W celu zorganizowania plików modułów w logiczną całość możemy zorganizować je w strukturę pakietów modułów. Jest ona oparta na hierarchii katalogów (folderów) systemu operacyjnego.

```
import mymodules.fibo
```

Kropka oznacza, że w podkatalogu mymodules znajdziemy moduł fibo. Katalog mymodules musi być umieszczony w jednym z katalogów znajdujących się na ścieżce wyszukiwania modułów (np. w zmiennej środowiskowej \$PYTHONPATH).

Aby katalogi były przeszukiwane podczas importu, muszą zawierać plik o nazwie `__init__.py`.

```
mycode\
  mymodules\
    __init__.py
      fibo.py
        fobo.py
```

W powyższym przykładzie, mycode nie jest pakietem.

Plik `__init__.py` może zawierać instrukcje, które zostaną automatycznie wykonane podczas importu pakietu.

Importowanie

Importowanie całego modułu

```
import os
```

```
import os
os

<module 'os' from '/usr/lib/python2.7/os.pyc'>

import os as different_name
different_name

<module 'os' from '/usr/lib/python2.7/os.pyc'>

os.getcwd()
'/home/chris/courses/py-2017-11-18'
```

Importowanie wybranych obiektów

```
from os import path
path

<module 'posixpath' from '/usr/lib/python2.7 posixpath.pyc'>

from os import path as different_name
different_name

<module 'posixpath' from '/usr/lib/python2.7 posixpath.pyc'>

from os import getcwd
getcwd()

'/home/chris/courses/py-2017-11-18'
```

Importowanie wszystkich obiektów (niezalecane)

```
from os import *
path

<module 'posixpath' from '/usr/lib/python2.7 posixpath.pyc'>

name
'posix'

getcwd()

'/home/chris/courses/py-2017-11-18'
```

sys.path

W momencie napotkania instrukcji importu, interpreter szuka modułu lub pakietu w:

- bieżącym katalogu,
- katalogach określonych w zmiennej systemowej \$PYTHONPATH
- w domyślnej ścieżce instalacji (np. /usr/local/lib/python)

Dokładną kolejność przeglądania katalogów określa sys.path:

```
import sys
sys.path

[",
'/usr/lib/python2.7',
'/usr/lib/python2.7/plat-x86_64-linux-gnu',
'/usr/lib/python2.7/lib-tk',
'/usr/lib/python2.7/lib-old',
'/usr/lib/python2.7/lib-dynload',
'/home/chris/.local/lib/python2.7/site-packages',
'/usr/local/lib/python2.7/dist-packages',
'/usr/lib/python2.7/dist-packages',
'/usr/lib/python2.7/dist-packages/PILcompat',
'/usr/lib/python2.7/dist-packages/gtk-2.0',
'/usr/lib/python2.7/dist-packages/wx-3.0-gtk2',
'/home/chris/.local/lib/python2.7/site-packages/ipython/extensions',
'/home/chris/.ipython']
```

Import vs wykonanie

Jak odróżnić import od wykonania?

Wewnątrz modułu jego nazwa (nazwa pliku bez rozszerzenia .py) jest dostępna pod globalną zmienną `__name__`, pod warunkiem że moduł jest importowany. W przeciwnym razie jego nazwa to "`__main__`".

Dlatego jeżeli chcemy wykonać kod tylko w momencie, gdy nasz plik jest wykonywany, a nie importowany, wówczas ten kod należy umieścić w ciele if `__name__ == '__main__'`:

virtualenv

[Link do dokumentacji](#)

Tworzenie nowego środowiska

W katalogu, w którym chcemy stworzyć nowy virtualenv należy wykonać poniższe polecenie:

```
virtualenv nazwa-katalogu
```

Powyzsze polecenie stworzy w bieżącym katalogu nowy katalog o nazwie nazwa-katalogu.

Określenie wersji pythona, która ma być wykorzystana w virtualenvie:

```
virtualenv nazwa-katalogu -p python2.7
```

Powyzsze polecenie zakłada, że python2.7 jest w katalogu wymienionym w \$PATH.

Używanie środowiska

Aktywacja środowiska polega m.in. na zmianie zmiennej \$PATH tak, aby wskazywała na Pythona z virtualenva, a nie na globalną instalację. Aktywacji trzeba dokonać w każdym terminalu osobno.

```
source katalog-virtualenvu/bin/activate
```

Deaktywacja środowiska:

```
deactivate
```

☆ Ćwiczenie

Stwórz dwa virtualenvy.

W pierwszym zainstaluj dowolną wersję Pythona 2 oraz bibliotekę flask w wersji 0.12.0.

W drugim zainstaluj dowolną wersję Pythona 2. Dodatkowo, zainstaluj tę samą bibliotekę co w pierwszym venvie, ale w wersji 0.12.1.

Przydatne polecenia:

```
python --version
pip freeze | grep flask
python -c "import flask"
python -c "import flask; print(json.__version__)"
```

Wyrażenia Regularne

API

```
import re
```

```
re.match
```

```
pattern = 'abc.e'  
re.match(pattern, 'abcde')
```

```
<_sre.SRE_Match at 0x7f6056f33440>
```

```
re.match(pattern, 'abcfe')  
<_sre.SRE_Match at 0x7f8cd8020100>
```

```
print(re.match(pattern, 'non-matching'))
```

```
None
```

re.search

```
assert re.match('cd', 'cd')  
assert re.match('cd', 'cdefg')  
assert not re.match('cd', 'abcd')
```

```
assert re.search('cd', 'cd')  
assert re.search('cd', 'cdefg')  
assert re.search('cd', 'abcd')
```

re.findall

```
m = re.search('[A-Z]{2}', 'aUUaUZaYUa')
```

```
m.start(), m.end()
```

```
(1, 3)
```

```
re.findall('[A-Z]{2}', 'aUUaUZaYUa')
```

```
['UU', 'UZ', 'YU']
```

Kompilacja wzorców

```
regex = re.compile(pattern)
```

```
regex.match('abcde')
```

```
<_sre.SRE_Match at 0x7f6056f33578>
```

```
regex.match('abcfe')
```

```
<_sre.SRE_Match at 0x7f8cd80207e8>
```

```
print(regex.match('non-matching'))
```

```
None
```

Flagi (re.VERBOSE)

```
pattern = ""  
    abc # search for "abc"  
    .   # then any character  
    e   # followed by "e" character  
..."  
re.match(pattern, 'abcfe', re.VERBOSE)
```

```
<_sre.SRE_Match at 0x7f8ccb792510>
```

```
regex = re.compile(pattern, re.VERBOSE)  
regex.match('abcfe')
```

```
<_sre.SRE_Match at 0x7f8ccb7923d8>
```

Nienazwane grupy

```
pattern = ""  
    abc # search for "abc"  
    (.) # then any character
```

```
    # then any character
e    # followed by "e" character
...
re.match(pattern, 'abcdef', re.VERBOSE).groups()
('f,)
```

Nazwane grupy

```
pattern = """
    abc      # search for "abc"
    (?P<char> .)  # then any character
    e        # followed by "e" character
...
re.match(pattern, 'abcdef', re.VERBOSE).groupdict()
{'char': 'f'}
```



```
re.match('(?P<dwie_litery>[A-Z]{2})+$', 'UUABCD').groupdict()
{'dwie_litery': 'CD'}
```

Składnia - Ściąga

Podstawy składni

Dowolny znak: .

```
assert re.match('abc.e', 'abcde')
assert re.match('abc.e', 'abfce')
assert not re.match('abc.e', 'abce')
assert not re.match('abc.e', 'abcdde')
```

Opcjonalnie: ?

```
assert re.match('abcd?e', 'abce')
assert re.match('abcd?e', 'abcde')
assert not re.match('abcd?e', 'abcfe')
assert not re.match('abcd?e', 'abcdde')
```

Powtórzenie n razy: {n}

```
assert not re.match('abcd{2}e', 'abce')
assert not re.match('abcd{2}e', 'abcde')
assert re.match('abcd{2}e', 'abcdde')
```

Grupowanie

```
assert re.match('ab(cd)?e', 'abe')
assert re.match('ab(cd)?e', 'abcde')
assert not re.match('ab(cd)?e', 'abfe')
assert not re.match('ab(cd)?e', 'abcdfe')
assert not re.match('ab(cd)?e', 'abce')
```

Alternatywa: |

```
assert re.match('ab(c|d)e', 'abce')
assert re.match('ab(c|d)e', 'abde')
assert not re.match('ab(c|d)e', 'abe')
assert not re.match('ab(c|d)e', 'abcde')
```

Grupa alternatyw: []

```
assert re.match('ab[cd]e', 'abce')
assert re.match('ab[cd]e', 'abde')
assert not re.match('ab[cd]e', 'abe')
```

```
assert not re.match('ab[cd]e', 'abcde')
```

Grupa znaków: [-]

```
assert re.match('ab[c-f]e', 'abce')
assert re.match('ab[c-f]e', 'abde')
assert re.match('ab[c-f]e', 'abfe')
assert not re.match('ab[c-f]e', 'abe')
assert not re.match('ab[c-f]e', 'abcde')
assert not re.match('ab[c-f]e', 'abcge')
```

```
assert re.match('ab[a-zA-Z0-9]e', 'abZe')
```

Negacja: [^]

```
assert not re.match('ab[^cd]e', 'abce')
assert not re.match('ab[^cd]e', 'abde')
assert not re.match('ab[^cd]e', 'abe')
assert not re.match('ab[^cd]e', 'abcde')
assert re.match('ab[^cd]e', 'abfe')
```

Escaping: \

```
assert re.match('ab\[d', 'ab[d')
assert not re.match('ab\[d', 'abcd')
```

Dopasuj do końca napisu: \$

```
assert re.match('abc', 'abc')
assert re.match('abc', 'abcd')
```

```
assert re.match('abc$', 'abc')
assert not re.match('abc$', 'abcd')
```

Dopasuj do początku napisu: ^

match zawsze dopasowuje do początku napisu:

```
assert re.match('bc', 'bc')
assert not re.match('bc', 'abc')
```

ale search już nie:

```
assert re.search('bc', 'bc')
assert re.search('bc', 'abc')
```

Chyba, że użyjemy ^:

```
assert re.search('bc', 'bc')
assert not re.search('^bc', 'abc')
```

Proste Regexpy

W każdym ćwiczeniu należy stworzyć zmienną patternX.

Następnie, skompiluj wzorzec i zapisz go w regexX. Przy komplikacji użyj flagi re.VERBOSE. Dodaj komentarze, aby zwiększyć czytelność kodu. Dodaj nazwane grupy tak, aby można było wyciągnąć określone dane.

Przykład

Rozwiążanie

Oczekiwane zachowanie

```
assert re.match(pattern, 'aba')
assert re.match(pattern, 'abb')
assert re.match(pattern, 'abz')
assert re.match(pattern, 'ab7')
assert re.match(pattern, 'ab[')
assert not re.match(pattern, 'ab')
assert not re.match(pattern, 'abcc')
```

```
assert regex.match('aba').groupdict() == {'char': 'a'}
assert regex.match('abz').groupdict() == {'char': 'z'}
assert regex.match('ab7').groupdict() == {'char': '7'}
assert regex.match('ab[').groupdict() == {'char': '['}
```

☆ Prosty pattern

Rozwiązanie

Oczekiwane zachowanie

```
assert re.match(pattern1, 'ab')
assert re.match(pattern1, 'abc')
assert re.match(pattern1, 'abd')
assert not re.match(pattern1, 'abcd')
assert not re.match(pattern1, 'abe')
```

```
assert regex1.match('ab').groupdict() == {'optional_group': ''}
assert regex1.match('abc').groupdict() == {'optional_group': 'c'}
assert regex1.match('abd').groupdict() == {'optional_group': 'd'}
```

☆ Numer telefoniczny

Rozwiązanie

Oczekiwane zachowanie

```
assert re.match(pattern2, '123 456 789')
assert re.match(pattern2, '678 543 970')
assert re.match(pattern2, '987654321') # grouping digits is optional
assert re.match(pattern2, '123-456-789') # you can separate groups by space or dash (or not separate it at all)
assert not re.match(pattern2, '12 345 678') # there must be exactly 9 digits
assert not re.match(pattern2, '12345678')
assert not re.match(pattern2, '1234567890')
assert not re.match(pattern2, '12 3456 789') # grouping matters (3 digits in each group)
```

```
assert regex2.match('123 456 789').groupdict() == {'group1': '123', 'group2': '456', 'group3': '789'}
assert regex2.match('987654321').groupdict() == {'group1': '987', 'group2': '654', 'group3': '321'}
```

☆ Numer telefoniczny z prefiksem państwa

Rozwiązanie

Oczekiwane zachowanie

```
assert re.match(pattern3, '123 456 789')
assert re.match(pattern3, '678 543 970')
assert re.match(pattern3, '123456789')
assert re.match(pattern3, '123-456-789')
assert re.match(pattern3, '+48 123 456 789')
assert re.match(pattern3, '+48 123456789')
```

```
assert re.match(pattern3, '+48123456789')
assert re.match(pattern3, '+1 345 111 222')
assert not re.match(pattern3, ' 345 111 222')
assert not re.match(pattern3, '12 456 789') # there must be exactly 9 digits (plus optional country prefix)
assert not re.match(pattern3, '12345678')
assert not re.match(pattern3, '123456789')
assert not re.match(pattern3, '12 3456 789') # grouping matters (3 digits in each group)
assert not re.match(pattern3, '+489 123 456 789') # one or two digits allowed after plus
assert not re.match(pattern3, '48 123 456 789') # if country prefix is present, it must be prepended by plus
assert not re.match(pattern3, '+123 456 789') # if plus is present, country prefix is required
```

```
assert regex3.match('123456789').groupdict() == {'prefix': None, 'group1': '123', 'group2': '456', 'group3': '789'}
assert regex3.match('+1 345 111 222').groupdict() == {'prefix': '1', 'group1': '345', 'group2': '111', 'group3': '222'}
```

☆ Adres email

Zakładamy, że w adresie email po małpie musi się pojawić dokładnie jedna kropka.

Rozwiązanie

Oczekiwane zachowanie

```
assert re.match(pattern4, 'john@smith.com')
assert re.match(pattern4, 'john.smith@gmail.com')
assert re.match(pattern4, 'j.o.h.n@smith.com')
assert re.match(pattern4, 'j.o.h.n+smith@smith.com')
assert re.match(pattern4, 'j.o.h.n+smi+th@smith.com')
assert not re.match(pattern4, 'jo@hn@smith.com')
assert not re.match(pattern4, 'jo@hn@com')
assert not re.match(pattern4, 'jo@hn@sm.ith.com')
```

```
assert regex4.match('john@smith.com').groupdict() == {'username': 'john', 'tag': None, 'host': 'smith.com', 'domain': 'com', 'subdomain': 'smith'}
assert regex4.match('j.o.h.n+tag@smith.com').groupdict() == {'username': 'j.o.h.n+tag', 'tag': 'tag', 'host': 'smith.com', 'domain': 'com', 'subdomain': 'smith'}
assert regex4.match('j.o.h.n+ta+g@smith.com').groupdict() == {'username': 'j.o.h.n+ta+g', 'tag': 'ta+g', 'host': 'smith.com', 'domain': 'com', 'subdomain': 'smith'}
```

☆ Data

Rozwiązanie

Oczekiwane zachowanie

```
assert re.match(pattern5, 'Sun Oct 14 13:47:03')
assert re.match(pattern5, 'Mon Dec 14 13:47:03')
assert not re.match(pattern5, 'Sun Oct 14 13:47:3')
assert not re.match(pattern5, 'Sun Aaa 14 13:47:03')
assert not re.match(pattern5, 'Sun Oct 40 13:47:03')
```

```
assert regex5.match('Sun Oct 14 13:47:03').groupdict() == {'day_of_week': 'Sun', 'month': 'Oct', 'day': '14', 'hour': '13', 'minute': '47', 'second': '03'}
```

☆ HTML

Napisz funkcję parse, która przyjmuje otwierający tag HTML jako napis i zwraca informacje o tym tagu.

Rozwiązanie

Oczekiwane zachowanie

```
assert parse('<html>') == ('html', {})
assert parse('invalid') == None
assert parse('<html key=value>') == ('html', {'key': 'value'})
```

```
assert parse('<num key= value >) == ('num', {'key': 'value'})
assert parse('<html key="value" key2=value2>') == ('html', {'key': 'value', 'key2': 'value2'})
assert parse('<html key="value" keyWithoutValue>') == ('html', {'key': 'value', 'keyWithoutValue':None})
```

Quantifiers

Zero, jeden lub więcej: *

```
assert re.match('abcd*e', 'abce')
assert re.match('abcd*e', 'abcde')
assert re.match('abcd*e', 'abcdde')
```

Jeden lub więcej: +

```
assert not re.match('abcd+e', 'abce')
assert re.match('abcd+e', 'abcde')
assert re.match('abcd+e', 'abcdde')
```

Zero lub jeden (fragment opcjonalny): ?

```
assert re.match('abcd?e', 'abce')
assert re.match('abcd?e', 'abcde')
assert not re.match('abcd?e', 'abcdde')
```

Dokładnie n razy: {n}

```
assert not re.match('abcd{2}e', 'abce')
assert not re.match('abcd{2}e', 'abcde')
assert re.match('abcd{2}e', 'abcdde')
```

Występujący od n do m razy: {n,m}

```
assert not re.match('abcd{2,4}e', 'abcde')
assert re.match('abcd{2,4}e', 'abcdde')
assert re.match('abcd{2,4}e', 'abcdde')
assert re.match('abcd{2,4}e', 'abcddee')
assert not re.match('abcd{2,4}e', 'abcddeee')
```

n lub więcej razy: {n,}

```
assert not re.match('abcd{2,}e', 'abcde')
assert re.match('abcd{2,}e', 'abcdde')
assert re.match('abcd{2,}e', 'abcdde')
assert re.match('abcd{2,}e', 'abcddee')
assert re.match('abcd{2,}e', 'abcddeee')
```

n lub mniej razy: {,n}

```
assert re.match('abcd{,4}e', 'abcde')
assert re.match('abcd{,4}e', 'abcdde')
assert re.match('abcd{,4}e', 'abcdde')
assert re.match('abcd{,4}e', 'abcddee')
assert not re.match('abcd{,4}e', 'abcddeee')
```

Klasy znaków

Cyfra: \d

```
assert re.match('\d', '0')
assert re.match('\d', '3')
assert not re.match('\d', 'a')
assert not re.match('\d', 'T')
assert not re.match('\d', '\n')
```

Nie cyfra: \D

```
assert not re.match('\D', '0')
assert not re.match('\D', '3')
assert  re.match('\D', 'a')
assert  re.match('\D', 'l')
assert  re.match('\D', '\n')
```

Biały znak \s

```
assert  re.match('\s', ' ')
assert  re.match('\s', '\t')
assert  re.match('\s', '\n')
assert not re.match('\s', 'a')
assert not re.match('\s', 'l')
assert not re.match('\s', '3')
```

Nie biały znak: \S

```
assert not re.match('\S', ' ')
assert not re.match('\S', '\t')
assert not re.match('\S', '\n')
assert  re.match('\S', 'a')
assert  re.match('\S', 'l')
assert  re.match('\S', '3')
```

Znaki specjalne: \n \t

```
assert  re.match('\n', '\n')
assert not re.match('\n', ' ')
assert not re.match('\n', '\t')
```

★ Apache Logs

Sparsuj jedną linię logów Apacha i zamień ją na słownik.

Rozwiązanie

Oczekiwane zachowanie

```
line = '64.242.88.20 - - [07/Mar/2017:16:24:48 -0800] "POST /login?next=/dashboard/ HTTP/1.0" 302 -'
```

```
pattern.match(line).groupdict()
```

```
{'content_length': '-',
'datetime': '07/Mar/2017:16:24:48 -0800',
'ip': '64.242.88.20',
'method': 'POST',
'protocol': 'HTTP/1.0',
'response_code': '302',
'url': '/login?next=/dashboard/'}
```

Dobre Praktyki w Pythonie

Code style [PEP8](#)

Najważniejsze rekommendacje PEP8 dotyczą:

- [wcieć](#),
- [sortowania importów](#),
- [spacji](#),
- [konwencji związanych z wielkością liter w nazwach klas, metod, funkcji, stałych itd](#)

- nazewnictwa,
- licznych niezwiązań ze sobą szczegółów.

pep8 tool

[pep8](#) to narzędzie konsolowe, które można zainstalować używając pip i które sprawdza poprawność kodu z PEP8. Rekomendowany sposób użycia:

```
pep8 --show-source --show-pep8 plik-do-sprawdzenia.py
```

pep8online.com

<http://pep8online.com> to darmowa internetowa usługa sprawdzająca poprawność kodu z PEP8.

★ Zastosuj PEP8 do apache2csv.py

★ Zastosuj PEP8 w pozostałych ćwiczeniach

Docstring Conventions [PEP257](#)

[setup.py](#)

requirements.txt

W pliku requirements.txt należy umieścić zależności dla projektu, tzn. wymagane biblioteki oraz opcjonalnie ich wersje. Biblioteki powinny być udostępnione w pypi. Instalacja wszystkich zależności jest wówczas prosta:

```
pip install -r requirements.txt
```

Powyższe polecenie zainstaluje także rekurencyjnie biblioteki, które są wymagane do działania bibliotek wymienionych w pliku requirements.txt.

Więcej o tym pliku można przeczytać w [dokumentacji pip'a](#).