Skrypty, słowniki, napisy

Parametry uruchomieniowe

Obiekt sys.arg:

Obiekt ‘argv’ w module ‘sys’ to lista **napisów** - parametrów wywołania skryptu.

Przykład - plik arg.py:

```
import sys
print(sys.argv)
for x in sys.argv:
    print(x)
```

Przykład – wywołanie:

```
py  arg.py 1 2 3 "wyraz2 wyraz2" '*'
```

```
['D:\\mwisniew_notatki\\python\\zajecia\\przyklady\\2_skr\\arg.py', '1', '2', '3', 'wyraz2 wyraz2', "'*'"]
D:\mwisniew_notatki\python\zajecia\przyklady\2_skr\arg.py
1
2
3
wyraz2 wyraz2
'*'
```

Słowniki

dict (słownik):

- tablica asocjacyjna (mapa, hash...,)
- klucz słownika to obiekt niemodyfikowalny (ang. immutuable), np. napis, liczba, krotka (np. lista lub słownik to obiekty modyfikowalne, ang. mutuable),
- wartością może być dowolny obiekt (nawet ten sam słownik),

Słowniki - przykłady

```
>>> #pusty słownik
...
>>> slownik={}
>>> #słownik z wartościami początkowymi:
...
>>> slownik={
... 'klucz':'wartosc',
... 'klucz2':[],
... 10:'dziesiec',
... (1,2,3):'krotka',
... }
>>> slownik
{'klucz': 'wartosc', 'klucz2': [], 10: 'dziesiec', (1, 2, 3): 'krotka'}
>>> # przypisanie wartości dla klucza o wartości 0:
...
>>> slownik[0]='zero'
>>> # nadpisanie wartości dla istniejącego klucza:
...
>>> slownik['klucz']='nowa wartosc'
>>> slownik
{'klucz': 'nowa wartosc', 'klucz2': [], 10: 'dziesiec', (1, 2, 3): 'krotka', 0:
'zero'}
>>>
```

Słowniki - przykłady

```
>>> slownik['self']=slownik
>>> slownik
{'klucz': 'nowa wartosc', 'klucz2': [], 10: 'dziesiec', (1, 2, 3): 'krotka', 0:
'zero', 'self': {...}}
>>> slownik['self']
{'klucz': 'nowa wartosc', 'klucz2': [], 10: 'dziesiec', (1, 2, 3): 'krotka', 0:
'zero', 'self': {...}}
>>>
```

Słowniki - przykłady

Próba pobrania wartość dla klucza, którego nie ma w słowniku powoduje wyrzucenie wyjątku ‘KeyError’.

```
>>> slownik['nie ma klucza']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'nie ma klucza'
>>>
```

Słowniki - przykłady

Gdy chemy uniknąć przechwytywania wyjątku, można użyć metody ‘get’.

```
>>> slownik.get('nie ma klucza')
>>> print(slownik.get('nie ma klucza'))
None
>>> print(slownik.get('nie ma klucza','wartość domyślna: brakuje klucza'))
wartość domyślna: brakuje klucza
>>>
```

Iterowanie po słowniku

Kolejnośc kluczy w słowniku jest przypadkowa.
Przykład iterowanie po kluczach:

```
>>> for x in slownik:
...     print(x)
...
klucz
klucz2
10
(1, 2, 3)
self
>>>
```

Iterowanie po słowniku

Podstawowe metody słownika:

- {}.keys() – zwraca obiekt z kluczami (tzw. view object),
- {}.values() – zwraca obiekt wartości,
- {}.items() – zwraca obiekt (klucz, wartość)

Przykład – iterowanie po kluczach i wartościach:

```
>>> for klucz, wartosc in slownik.items():
...     print(klucz, wartosc)
...
klucz wartość
klucz2 []
10 dziesięć
(1, 2, 3) ddd
self {...}
>>>
```

Napisy - inicjowanie

Napisy - obiekty klasy str; przykłady tworzenia obiektów str:

```
>>> nap1='przykład'
>>> nap1='przykład1'
>>> nap2="przykład2"
>>> nap3=" przykład3 'cyt' "
>>> nap4=' przykład3 'cyt' '
  File "<stdin>", line 1
    nap4=' przykład3 'cyt' '
                     ^
SyntaxError: invalid syntax
>>> nap4=' przykład3 "cyt" '
>>> nap5="""przykład5, linia1
... linia2,
... linia3"""
>>>
```

Napisy – konkatenacja

Napisy są niemodyfikowalne. Przykład – za każdym przejściem pętli tworzony jest nowy obiekt ‘napis’:

```
>>> lista_napisow=[
... 'abc','def','ghi','jkl','mno','pqr'
... ]
>>> wynik=''

>>> for napis in lista_napisow:
...     wynik+=napis
...
>>> wynik
'abcdefghijklmnopqr'
>>>
```

Napisy – metoda join()

Przykład łączenia napisów metodą join():

```
>>> lista_napisow
['abc', 'def', 'ghi', 'jkl', 'mno', 'pqr']
>>> print(''.join(lista_napisow))
abcdefghijklmnopqr
>>> '.'.join(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k'])
'a.b.c.d.e.f.g.h.i.j.k'
>>>
```

Metoda .join() powoduje użycie napisu jako separatora do połączenia wszystkich napisów w liście.

Napisy – metoda split()

Metoda split():

```
>>> 'a b c'.split()
['a', 'b', 'c']
>>> 'aa/bb/cc'.split('/')
['aa', 'bb', 'cc']
>>>
```

Pliki tekstowe

Aby otworzyć plik, używamy wbudowanej funkcji open(), która przyjmuje nazwę pliku i tryb otwarcia. Jeśli podamy tylko nazwę pliku, zostanie on otwarty w trybie tekstowym i tylko do odczytu.

```
#I wersja
f = open("plik.txt")
print(f.read())
f.close()

#II druga wersja
f = open("plik.txt")
for line in f:
    print(line)
f.close()

#III wersja
with open("plik.txt") as f:
    print(f.read())
#tutaj plik jest zamknięty
```

Pliki tekstowe – metoda read()

metoda read() – wczytanie całej zawartości

```
f=open('example.txt','rt')
calosc=f.read()
print(calosc)
f.close()
```

metoda read(size) – czytanie po znaku

```
f=open('example.txt','rt')
ch=f.read(1)
while ch!='':
    print(ch, end='')
    ch=f.read(1)
f.close()
```

Pliki tekstowe – metoda seek()

metoda seek(offset, [whence]): whence=0 – względem początku pliku (domyślne), whence=1 względem aktualnej pozycji, whence=2 – względem końca pliku (w trybie tekstowym opcje whence mają ograniczone działanie)

```
f=open('example.txt','rt')
some=f.read(7)
print(some)
f.seek(0,0)     #idź do początku
some=f.read(7)
print(some)
print(f.tell())
f.seek(0,2)       #idź do końca
print(f.tell())
f.seek(0,0)
print(f.tell())
f.close()
```

Pliki tekstowe – metoda readline()

metoda readline(size) – czyta znaki od aktualnej pozycji do końca linii lub ‚size' znaków do końca linii:

```
f=open('example.txt','rt')
line=f.readline()
print(line,end='')
f.seek(55,0)
line=s.readline()
print(line,end='')
f.seek(58,0)
line=f.readline(4)
print(line,end='')
f.close()
```

Pliki i wyjątki

Pliki - obsługa wyjatków:

```
from os import strerror

#Przykład 1:
try:
    s=open('1234.txt','rt')
    print(s.read())
    s.close()
except:
    print('blad')

#Przykład 2:
try:
    s=open('1234.txt','rt')
    print(s.read())
    s.close()
except IOError as e:
    print("I/O error occured: ", strerror(e.errno))
```

Zadanie 1

Napisz program, który przyjmuje w linii poleceń jeden parametr — liczbę całkowitą i wypisuje komunikat na temat jej parzystości.

Zadanie 2

Napisz funkcję, która przyjmuje jako parametr napis w formacie:

'k1:v1,k2:v2,k3:v3'

i zwraca słownik o zawartości:

{'k1': 'v1', 'k2': 'v2', 'k3': 'v3'}

Zadanie 3

Napisz funkcję, która przyjmuje jako parametr napis w formacie:

'''k1: v1

k2: v2

k3: v3'''

i zwraca słownik o zawartości:

{'k1':'v1','k2':'v2','k3':'v3'}

Zadanie 4

Utwórz plik tekstowy „plik4.txt” o zawartości:

k1:v1

k2:v2

k3:v3

Następnie napisz program, który przyjmuje jako parametr nazwę klucza (k1,k2 lub k3) i wypisuje na wyjściu odpowiednią wartość.

Zadanie 5

Napisz program, który przyjmuje w linii poleceń dwa parametry:

1) nazwa pliku lub znak **-** , 2) szukane słowo.

Program ma otwierać plik, którego nazwę podano jako pierwszy parametr lub użyć funkcji **readlines()** z modułu **sys.stdin**, jeśli podano znak **-** . Program ma wyświetlić wszystkie linie podane na wejściu, które zawierają podane słowo.

Zadanie 6

Wydrukuj na ekranie alfabet w następujący sposób: aAbBcC…

Zadanie 7

Zmodyfikuj poprzedni program tak aby wyświetlał co n-tą parę liter (małaDUŻA) z alfabetu. Liczbę n wprowadza użytkownik (jest pytany przez program, wykorzystaj wbudowaną funkcje input()).

Zadanie 8

Napisz program szyfrujący podany ciąg znaków szyfrem Cezara (tylko małe litery alfabetu).

Zadanie 9

Napisz program który w parametrze otrzymuje dwie nazwy plików. Program ma wczytać jeden plik, zaszyfrować go szyfrem Cezara i zapisać pod nową nazwą (2-gi parametr).