Politechnika Wrocławska

Python
wstęp do programowania
dla użytkowników WCSS

Dr inż. Krzysztof Berezowski
Instytut Informatyki, Automatyki i Robotyki
Politechniki Wrocławskiej

Wejście/wyjście

OPERACJE NA PLIKACH

Obiekt file

- Funkcja open tworzy obiekt związany z plikiem

```
>>> f = open("C:/…/listaSluchaczy_E10-81r.csv")
>>> print type(f)
<type ’file’>
```

- Wczytanie całego pliku

```
>>> f = open("C:/…/listaSluchaczy_E10-81r.csv")
>>> f.read()
‘'\nPolitechnika Wroc\xb3awska\nRok akademicki;2010/2011\nTyp
    kalendarza;dwusemestralny…’
```

- Wczytanie jednej linii

```
>>> f = open("C:/…/listaSluchaczy_E10-81r.csv")
>>> f.readline()
'Politechnika Wroc\xb3awska\n'
```

Obiekt file

- Funkcja open tworzy obiekt związany z plikiem

```
>>> f = open("C:/…/listaSluchaczy_E10-81r.csv")
>>> print type(f)
<type ’file’>
```

- Wczytanie całego pliku

```
>>> f = open("C:/…/listaSluchaczy_E10-81r.csv")
>>> f.read()
‘'\nPolitechnika Wroc\xb3awska\nRok akademicki;2010/2011\nTyp
    kalendarza;dwusemestralny…’
```

- Wczytanie jednej linii

```
>>> f = open("C:/…/listaSluchaczy_E10-81r.csv")
>>> f.readline()
'Politechnika Wroc\xb3awska\n'
```

Plik jako zbiór linii

- Wczytanie pliku jako listy linii

```
>>> f = open("C:/…/listaSluchaczy_E10-81r.csv")
>>> f.readlines()
['\n', 'Politechnika Wroc\xb3awska\n', 'Rok
     akademicki;2010/2011\n', …]
```

- Przetwarzanie pliku po linii

```
>>> f = open("C:/…/listaSluchaczy_E10-81r.csv")
>>> for line in f:
        print line
…
```

Najważniejsze tryby funkcji open

Tryb	Interpretacja algebraiczna	Uwagi
`r`	Odczyt tekstowy	Tylko istniejące pliki. Wskaźnik pliku na początku pliku. Pliki binarne od tekstowych różnią się przetwarzaniem znaku nowej linii.
`rb`	Odczyt binarny	
`r+`	Odczyt z zapisem	
`rb+`	Odczyt binarny z zapisem	
`w`	Zapis	Otwiera istniejący bądź tworzy nowy plik. Wskaźnik pliku na początku pliku.
`wb`	Zapis binarny	
`w+`	Zapis z odczytem	
`wb+`	Zapis binarny z odczytem	
`a`	Dołączanie	Dopisuje do istniejącego bądź tworzy nowy plik. Wskaźnik pliku na końcu pliku.
`ab`	Dołączanie binarne	
`a+`	Dołączanie z odczytem	
`ab+`	Dołączanie binarne z odczytem	

Zamykanie pliku

- Otwarte pliki są zamykane automatycznie przy usunięciu obiektu `file`
- W niektórych przypadkach może to wyczerpać zasoby uchwytów plikowych
- W rezultacie należy pamiętać o zamykaniu
- Albo używać klauzuli with

```
with open("C:/.../listaSluchaczy_E10-81r.csv") as f:
    for line in f:
        print line
```

TU MA BYĆ PRZYKŁAD ☺

Organizacja przepływu sterowania

FUNKCJE

Definicja funkcji

- Definicja funkcji

```
>>> def f():
     print "Hello, World!"
                                    '
>>> print type(f)
<type 'function'>
>>> print f()
Hello, World!
None
>>>
```

- Funkcja jest obiektem
- Funkcja zawsze zwraca wartość.

Wejście i wyjście funkcji

```
>>> def welcome(who):
    print "Hello, %s!" % who
    return who == "World"

>>> print welcome("World")
Hello, World!
True
>>> print welcome("Bob")
Hello, Bob!
False
>>>
```

- Funkcja może:
  - przyjmować argumenty wejściowe
  - zwracać wartości (obliczone z argumentów)
  - wykonywać inne czynności uboczne

Komunikacja „z funkcją”

```
>>> def welcome(what, who):
    print "%s, %s!" % (what, who)
    return who == "World", what == "Hello"

>>> print welcome("Hello","World")
Hello, World!
(True, True)
>>> print welcome("Hi", "Bob")
Hi, Bob!
(False, False)
>>>
```

- Dowolna skończona liczba argumentów formalnych
- Dowolna skończona liczba wartości wyjściowych
  (funkcja konstruuje krotkę wyjściową)

Argumenty domyślne

```
>>> def welcome(what="Hello", who="World"):
    print "%s, %s!" % (what, who)
    return who == "World", what == "Hello"

>>> print welcome()
Hello, World!
(True, True)
>>> print welcome("Hi")
Hi, World!
(True, False)
>>> print welcome("Hi", "Bob")
Hi, Bob!
(False, False)
```

- Argumenty z wartościami domyślnymi mogą być pomijane przy wywołaniu funkcji
- Argumenty bez wartości domyślnych są zawsze z przodu

Argumenty przekazywane jawnie

```
>>> def welcome(what="Hello", who="World"):
    print "%s, %s!" % (what, who)
    return who == "World", what == "Hello"

>>> print welcome(who="Bob")
Hello, Bob!
(False, True)

>>> print welcome("Yo", who=„Susan")
Yo, Susan!
(False, False)
```

- Argumenty przekazywane jawnie w dowolnym porządku
- Argumenty nienazwane zawsze przed nazwanymi
- Wiązanie nienazwanych: pozycyjne
- Wiązanie nazwanych: po nazwie

Argumenty dowolne * i **

```
>>> def funkcja(*k):
    print k
>>> funkcja("Hi", "Bob")
('Hi', 'Bob')
```

- Pozwala na przekazanie dowolnej ilości argumentów do funkcji (listy argumentów)

```
>>> def funkcja(**k):
    print k
>>> funkcja(what="Hi", who="Bob")
{'what': 'Hi', 'who': 'Bob'}
```

- Pozwala na przekazanie dowolnej listy nazwanych argumentów (wymusza nazwane wywołanie).

Składnia mieszana

```
>>> def funkcja(a, b, c, *d, **e):
     print a, b, c
     print d
     print e.values()
>>> funkcja("tekst", 2, 3, 4, 5, 6, nazwisko="Kowalski")
Text 2 3                          '
(4, 5, 6)
['Kowalski']
```

- Wszystkie jawne parametry są wypełniane
- Kolejne nienazwane wstawiane do listy d
- Kolejne nazwane do słownika e
- Nazwane muszą być na końcu listy

Rozpakowywanie argumentów

- Listy oraz słowniki można rozpakować do przekazania do funkcji wieloargumentowej

```
>>> def funkcja(a, b, c, d):
     print a, b, c, d
>>> funkcja(1, 2, 3, 4)
1 2 3 4

>>> l = [1, 2, 3, 4]
>>> funkcja(l)
TypeError: funkcja() takes exactly 4 arguments (1 given)

>>> funkcja(*l)
1 2 3 4
```

- Analogicznie działają słowniki

Przykład

```
def mediana(*val):
    if len(val) > 0:
        v = sorted(val)
        if len(v) % 2 == 1:
            return v[len(v)/2]
        else:
            return (v[len(v)/2-1] + v[len(v)/2])/2.0

# mediana argumentow prostych
print mediana(1, 2, 3, 8)
print mediana(1, 3, 4, 3, 8)

# mediana argumentow podanych z rozpakowanej listy
l = [ 10, 33, 4, 1 ]
print mediana(*l)
```

Funkcja jako obiekt

```
>>> def funkcja(a, b, c, d):
      print a, b, c, d
>>> funkcja(1, 2, 3, 4)
1 2 3 4

>>> def wywolaj(f, *a):
      print type(f)
      return f(*a)

>>> wywolaj(funkcja, 3, 4, 5, 6)
<type ‘function’>
3 4 5 6

>>> wywolaj(welcome, ”Hi”) # z poprzednich slajdów
<type ‘function’>
‘Hi, World!’
```

- Funkcja jest elementem (obiektem) na którym mogą operować inne funkcje

Nienazwane formy lambda

```
>>> def wywolaj(f, *a):
      print type(f)
      return f(*a)

>>> wywolaj(lambda x, y: x+y, 3, 4)
<type ‘function’>
7
```

- Formy lambda pozwalają na konstruowanie obiektów typu funkcje „w miejscu” bez ich definiowania

- (Pokazać kilka przykładów ☺)