

# Wstęp do programowania

## Operacje na plikach i napisy

Paweł Daniluk

Wydział Fizyki

Jesień 2013



# Wejście/wyjście

Większość problemów polega na przetwarzaniu danych. Trzeba je jakoś wczytać. Wygenerowane dane trzeba jakoś zapisać. Konieczna jest również możliwość interakcji z użytkownikiem.

## Wariant najprostszy

- Wczytywanie i zapisywanie plików tekstowych
- Standardowe wejście i wyjście (przetwarzanie potokowe)

# Wejście/wyjście

Do tej pory używaliśmy instrukcji `print` i `input`.

## Przykład

```
>>> liczba=input("Podaj liczbe:")  
Podaj liczbe:42  
>>> print "Liczba:", liczba  
Liczba: 42  
>>>
```

## Zamiana wartości na napis

Funkcja `str()` zamienia wartość na napis czytelny dla człowieka. Jest wywoływana niejawnie, gdy zachodzi potrzeba (np. w instrukcji `print`).

### Przykład

```
>>> str(2)
'2'
>>> str("Ala ma kota")
'Ala ma kota'
>>> str(range(3))
'[0, 1, 2]'
>>>
```

## Zamiana wartości na napis c.d.

Funkcja `repr()` zamienia wartość na napis interpretowalny w Pythonie.

### Przykład

```
>>> repr(2)
'2'
>>> repr("Ala ma kota")
"'Ala ma kota'"
>>> repr(range(3))
'[0, 1, 2]'
>>>
```

## Ładniejsze napisy

### Wyrównywanie tekstu

`str.ljust(width[, fillchar])` wyrównywanie do lewej

`str.center(width[, fillchar])` wyrównywanie do środka

`str.rjust(width[, fillchar])` wyrównywanie do prawej

### Przykład

```
>>> for i in range(1,10):
...     s=str(8**i)
...     print s.ljust(10),s.center(10),s.rjust(10), zfill(10)
...
8           8           8 0000000008
64          64          64 0000000064
512         512         512 0000000512
4096        4096        4096 0000004096
32768       32768       32768 0000032768
262144      262144      262144 0000262144
2097152     2097152     2097152 0002097152
16777216    16777216    16777216 0016777216
134217728   134217728   134217728 0134217728
```

## Ładniejsze napisy c.d.

### Formatowanie napisów

Operator % pozwala na wypełnianiu napisu treścią.

### Składnia

```
format % values
```

### Format – konwersje

- 1 znak '%',
- 2 (opcjonalnie) klucz w nawiasach, np. (somename),
- 3 (opcjonalnie) opcje konwersji (#, 0, -, ' ', +),
- 4 (opcjonalnie) minimalna szerokość,
- 5 (opcjonalnie) precyzja (po znaku '.')
- 6 conversion type (m.in. d – liczba całkowita, f – liczba zmiennoprzecinkowa, c – znak, s – napis)

## Ładniejsze napisy c.d.

### Przykłady

```
>>> print '%d / %d = %2.3f' % (2,3,2.0/3.0)
2 / 3 = 0.667
>>> print '%(language)s has %(number)03d quote types.' % \
... "language": "Python", "number": 2
Python has 002 quote types.
```



## Ładniejsze napisy c.d.

### Formatowanie napisów (nowa składnia)

Napisy mają metodę format.

### Przykłady

```
>>> print 'We are the {} who say "{}!"'.format('knights', 'Ni')
We are the knights who say "Ni!"
>>> print '{0} and {1}'.format('spam', 'eggs')
spam and eggs
>>> print '{1} and {0}'.format('spam', 'eggs')
eggs and spam
>>> print 'This {food} is {adjective}.'.format(
...     food='spam', adjective='absolutely horrible')
This spam is absolutely horrible.
>>> import math
>>> print 'The value of PI is approximately {0:.3f}'.format(math.pi)
The value of PI is approximately 3.142.
```

# Pliki

Dostęp do pliku odbywa się poprzez uchwyty, które są tworzone podczas otwarcia pliku i niszczone po jego zamknięciu. W Pythonie uchwyty do pliku zrealizowane są przy pomocy obiektów.

## Otwieranie i zamykanie

- `open(filename, mode)` – otwiera plik w trybie określonym parametrem
- `f.close()` – zamyka plik

## Tryby

- `'r'` – odczyt
- `'w'` – zapis
- `'a'` – zapis na końcu
- `'r+'` – odczyt i zapis

# Odczyt/zapis

## Odczyt

- `f.read(size)` – `size` znaków, lub cały plik
- `f.readline()` – jedna linia
- `f.readlines()` – wszystkie linie (w liście)

## Zapis

- `f.write(string)` – zapisuje znaki do pliku

## Położenie

- `f.tell()` – aktualne położenie w pliku
- `f.seek(offset, from_what)` – ustawia położenie względem (początku (domyślnie) – 0, bieżącej pozycji – 1, końca – 2)

## Iterowanie po liniach w pliku

```
>>> for line in f:  
    print line,
```

```
This is the first line of the file.  
Second line of the file
```

## Standardowe wejście

Programy w systemach zgodnych z POSIX (i nie tylko) mają trzy standardowe strumienie:

- standardowe wejście (`sys.stdin`)
- standardowe wyjście (`sys.stdout`)
- standardowy błąd (`sys.stderr`)

Służą one między innymi do zgrabnego implementowania przetwarzania potokowego.

### Przetwarzanie potokowe – przykład

```
pawel@hydra:~/tmp$ ls -Al | grep May | cut -c 42-  
May 10 2013 find_contacts.py  
May 6 2013 libusb-1.0.16-rc9  
May 6 2013 pyusb-1-1.0.0a3
```

Aby z nich korzystać trzeba zaimportować moduł `sys`.

# Operacje na napisach

## Konkatenacja – operator +

```
>>> s1,s2,s3=("Ala","ma","kota")
>>> s1+s2+s3
'Alamakota'
>>>
```

# Operacje na napisach c.d.

## Wyszukiwanie

`str.find(sub[, start[, end]])` znajduje początek pierwszego wystąpienia `sub` w `str[start:end]`

`str.rfind(sub[, start[, end]])` znajduje początek ostatniego wystąpienia `sub` w `str[start:end]`

`str.count(sub[, start[, end]])` znajduje liczbę rozłącznych wystąpień `sub` w `str[start:end]`

`str.startswith(prefix[, start[, end]])` sprawdza czy `prefix` występuje na początku `str[start:end]`

`str.endswith(suffix[, start[, end]])` sprawdza czy `suffix` występuje na końcu `str[start:end]`

`str.replace(old, new[, count])` zwraca ciąg otrzymany przez zastąpienie co najwyżej `count` wystąpień napisu `old` przez `new`

## Operacje na napisach c.d.

```
str='ala Ma kota'
```

### Wielkość znaków

```
str.capitalize() 'Ala ma kota'
```

```
str.lower() 'ala ma kota'
```

```
str.upper() 'ALA MA KOTA'
```

```
str.title() 'Ala Ma Kota'
```

```
str.swapcase() 'ALA mA KOTA'
```



## Operacje na napisach c.d.

### Testy

`str.isalnum()` znaki alfanumeryczne

`str.isalpha()` znaki z alfabetu

`str.isdigit()` cyfry

`str.islower()` wszystkie litery są małe

`str.isspace()` białe znaki

`str.istitle()` wielkie litery na początku słów, pozostałe małe

`str.isupper()` wszystkie litery są wielkie

## Operacje na napisach c.d.

### Obcinanie białych znaków

`str.strip([chars])` obcina znaki z `chars` (domyślnie białe) z obu stron `str`

`str.lstrip([chars])` obcina znaki z `chars` (domyślnie białe) z lewej strony `str`

`str.rstrip([chars])` obcina znaki z `chars` (domyślnie białe) z prawej strony  
`str`

## Operacje na napisach c.d.

### Cięcie napisów

`str.partition(sep)` podział napisu na trzy części wokół pierwszego wystąpienia `sep`

`str.rpartition(sep)` podział napisu na trzy części wokół ostatniego wystąpienia `sep`

`str.split([sep[, maxsplit]])` podział napisu na `maxsplit` części (od lewej)

`str.rsplit([sep[, maxsplit]])` podział napisu na `maxsplit` części (od prawej)

## Operacje na napisach c.d.

### Przykłady

```
>>> '1,,2'.partition(',')
('1', ', ', ',2')
>>> '1,,2'.rpartition(',')
('1,', ', ', '2')
>>> '1,,2'.split(',')
['1', '', '2']
>>> ' 1 2 3 '.split()
['1', '2', '3']
```

## Operacje na napisach c.d.

### Łączenie napisów

`sep.join(iterable)` łączy elementy listy przedzielając je napisem `sep`

### Przykład

```
>>> '<>'.join([str(i) for i in range(3)])  
'0<>1<>2'
```

## Argumenty wywołania

Lista `sys.argv` zawiera argumenty wywołania programu.

```
test.py
```

```
#!/usr/bin/python
```

```
import sys
```

```
print sys.argv
```

```
pawel@hydra:~/tmp$ ./test.py arg1 arg2 3 arg4  
['./test.py', 'arg1', 'arg2', '3', 'arg4']  
pawel@hydra:~/tmp$ ./test.py "arg1 arg2 3 arg4"  
['./test.py', 'arg1 arg2 3 arg4']  
pawel@hydra:~/tmp$
```

## Zadanie 1 – cat i tee

Zaimplementuj uproszczone wersje Uniksowych programów cat i tee.

### cat

Wywołanie:

```
cat.py nazwa_pliku
```

Program otwiera plik i wysyła jego zawartość na standardowe wyjście.

### tee

Wywołanie:

```
tee.py nazwa_pliku
```

Program zapisuje w pliku zawartość podaną na standardowe wejście.

## Zadanie 2

Zaimplementuj uproszczoną wersję programu `grep`.

Wywołanie:

```
grep.py tekst
```

Program wypisuje na standardowe wyjście numery i zawartość linii podanych na standardowe wejście, które zawierają podany tekst.



## Zadanie 3

Napisz program wypisujący na standardowe wyjście ładnie sformatowaną tabliczkę mnożenia o podanym rozmiarze.

## Zadanie 4

Zaimplementuj uproszczoną wersję programu cut.

Wywołanie:

```
cut.py separator kol1 kol2 ... kolN
```

Program wypisuje na standardowe wyjście podane kolumny ze standardowego wejścia. Pusty separator (”) oznacza dowolny ciąg białych znaków.

## Zadanie 5

Zaimplementuj uproszczoną wersję programu `join`.

Wywołanie:

```
join.py separator plik1 plik2 ... plikN
```

Program wypisuje na standardowe wyjście połączone separatorem kolumny z podanych plików.

[http://bioexploratorium.pl/wiki/Wstę\\_p\\_do\\_programowania\\_  
\\_2013z](http://bioexploratorium.pl/wiki/Wst%C4%99p_do_programowania_2013z)