# Wstęp do programowania

## Listy i zbiory
## Operacje na plikach
## Napisy

Paweł Daniluk

Wydział Fizyki

Jesień 2014

# Listy

## W poprzednich odcinkach...

```
>>> lista=[1,2,3,4,5]
>>> lista
[1, 2, 3, 4, 5]
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> lista[3]=8
>>> lista[1:4]
[2, 3, 8]
>>>
```

# Listy – metody

Python jest językiem obiektowym. Obiekty mogą mieć metody. Listy są obiektami.

## Wstawianie

list.append(x) Add an item to the end of the list; equivalent to `a[len(a):] = [x]`.

list.extend(L) Extend the list by appending all the items in the given list; equivalent to `a[len(a):] = L`.

list.insert(i, x) Insert an item at a given position. The first argument is the index of the element before which to insert, so `a.insert(0, x)` inserts at the front of the list, and `a.insert(len(a), x)` is equivalent to `a.append(x)`.

# Listy – metody

## Usuwanie elementów

| | |
|---|---|
| list.remove(x) | Remove the first item from the list whose value is `x`. It is an error if there is no such item. |
| list.pop(i) | Remove the item at the given position in the list, and return it. If no index is specified, `a.pop()` removes and returns the last item in the list. |

| | |
|---|---|
| list.index(x) | Return the index in the list of the first item whose value is `x`. It is an error if there is no such item. |
| list.count(x) | Return the number of times x appears in the list. |
| list.sort() | Sort the items of the list, in place. |
| list.reverse() | Reverse the elements of the list, in place. |
| len(list) | List length. |

# Listy – metody

## Przykłady

```
>>> a = [66.25, 333, 333, 1, 1234.5]
>>> print a.count(333), a.count(66.25), a.count('x')
2 1 0
>>> a.insert(2, -1)
>>> a.append(333)
>>> a
[66.25, 333, -1, 333, 1, 1234.5, 333]
>>> a.index(333)
1
>>> a.remove(333)
>>> a
[66.25, -1, 333, 1, 1234.5, 333]
>>> a.reverse()
>>> a
[333, 1234.5, 1, 333, -1, 66.25]
>>> a.sort()
>>> a
[-1, 1, 66.25, 333, 333, 1234.5]
```

# List comprehensions

Mając metodę `append` łatwo można produkować listy przy użyciu pętli `for`.

```
>>> squares = []
>>> for x in range(10):
...     squares.append(x**2)
...
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

# List comprehensions

Mając metodę `append` łatwo można produkować listy przy użyciu pętli `for`.

```
>>> squares = []
>>> for x in range(10):
...     squares.append(x**2)
...
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

To nie jest najładniejsze rozwiązanie.

```
squares = [x**2 for x in range(10)]
```

# List comprehensions c.d.

W *list comprehensions* można używać dowolnej kombinacji instrukcji `for` i `if`.

```
>>> [(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

Zamiast

```
>>> combs = []
>>> for x in [1,2,3]:
...     for y in [3,1,4]:
...         if x != y:
...             combs.append((x, y))
...
>>> combs
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

# Krotki (ang. *tuples*)

Krotki są jak listy typem sekwencyjnym. W odróżnieniu od list są niezmienne. Zapisywane są w nawiasach okrągłych.

```
>>> t = 12345, 54321, 'hello!'
>>> t[0]
12345
>>> t
(12345, 54321, 'hello!')
>>> # Tuples may be nested:
... u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
```

# Krotki (ang. *tuples*) c.d.

```
>>> # Tuples are immutable:
... t[0] = 88888
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> # but they can contain mutable objects:
... v = ([1, 2, 3], [1, 2, 3])
>>> v[1].reverse()
>>> v
([1, 2, 3], [3, 2, 1])
```

# Krotki (ang. *tuples*) c.d.

## Krotka pusta i jednoelementowa

```
>>> empty = ()
>>> singleton = 'hello',    # <-- note trailing comma
>>> len(empty)
0
>>> len(singleton)
1
>>> singleton
('hello',)
```

## Rozpakowywanie krotek i innych sekwencji

```
>>> t = 12345, 54321, 'hello!'
>>> x, y, z = t
```

# Napisy też są sekwencjami

```
>>> napis="Ala␣ma␣kota."
>>> napis[4:]
'ma␣kota.'
>>> napis[:3]
'Ala'
>>>
```

## Konkatenacja (łączenie) napisów

```
>>> napis[:-1]+"␣w␣glowie."
'Ala␣ma␣kota␣w␣glowie.'
```

## Powtarzanie napisów

```
>>> dzwon="Bim␣Bam␣Bom"
>>> (dzwon + '␣') * 3
'Bim␣Bam␣Bom␣Bim␣Bam␣Bom␣Bim␣Bam␣Bom␣'
```

# Operacje działające na wszystkich sekwencjach

| | |
|---|---|
| x in s | `True` if an item of `s` is equal to `x`, else `False` |
| x not in s | `False` if an item of `s` is equal to `x`, else `True` |
| s + t | the concatenation of `s` and `t` |
| s * n, n * s | `n` shallow copies of `s` concatenated |
| s[i] | `i`th item of `s`, origin 0 |
| s[i:j] | slice of `s` from `i` to `j` |
| s[i:j:k] | slice of `s` from `i` to `j` with step `k` |
| len(s) | length of `s` |
| min(s) | smallest item of `s` |
| max(s) | largest item of `s` |
| s.index(i) | index of the first occurrence of `i` in `s` |
| s.count(i) | total number of occurrences of `i` in `s` |

# Zbiory

Zbiory nie mogą zawierać powtórzeń.

```
>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'ban
>>> fruit = set(basket)
# create a set without duplicates
>>> fruit
set(['orange', 'pear', 'apple', 'banana'])
>>> 'orange' in fruit
# fast membership testing
True
>>> 'crabgrass' in fruit
False
```

# Zbiory c.d.

## Algebra zbiorów

```
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a                                    # unique letters in a
set(['a', 'r', 'b', 'c', 'd'])
>>> a - b
# letters in a but not in b
set(['r', 'd', 'b'])
>>> a | b
# letters in either a or b
set(['a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'])
>>> a & b
# letters in both a and b
set(['a', 'c'])
>>> a ^ b
# letters in a or b but not both
set(['r', 'd', 'b', 'm', 'z', 'l'])
```

# Zbiory c.d.

## Set comprehensions

```
>>> a = {x for x in 'abracadabra' if x not in 'abc'}
>>> a
set(['r', 'd'])
```

# Słowniki

Do elementów sekwencji odwołujemy się przez podanie indeksu. Słowniki są uogólnieniem tej notacji. Do elementów słownika odwołujemy się przez podanie wartości klucza.

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'sape': 4139, 'guido': 4127, 'jack': 4098}
>>> tel['jack']
4098
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'guido': 4127, 'irv': 4127, 'jack': 4098}
>>> tel.keys()
['guido', 'irv', 'jack']
>>> tel.values()
[4127, 4127, 4098]
>>> 'guido' in tel
```

# Słowniki c.d.

## Tworzenie przy pomocy konstruktora

```
>>> dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
{'sape': 4139, 'jack': 4098, 'guido': 4127}
>>>
>>> # When the keys are simple strings, it is sometimes easier t
...
>>> dict(sape=4139, guido=4127, jack=4098)
{'sape': 4139, 'jack': 4098, 'guido': 4127}
```

## Tworzenie przy pomocy *dictionary comprehension*

```
>>> {x: x**2 for x in (2, 4, 6)}
{2: 4, 4: 16, 6: 36}
```

# 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)

# Odczyt/zapis c.d.

## 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

Stwórz listę o zadanej długości zawierającą:

1. Kolejne liczby naturalne.
2. Kolejne liczby nieparzyste.
3. Kolejne liczby Fibonacciego.
4. Losowe wartości ze zbioru $\{0, 1\}$.

# Zadanie 2

Stwórz kwadratową tablicę o zadanym rozmiarze zawierającą tabliczkę mnożenia.

# Zadanie 3

Policz wystąpienia poszczególnych liter w zadanym słowie.

# Zadanie 3

Policz wystąpienia poszczególnych liter w zadanym słowie.

Jak można byłoby to zrobić, gdyby nie było metody `count`?

# Zadanie 4 – 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 5

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 6

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

# Zadanie 7

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 8

Zaimplementuj uproszczoną wersję programu `join`.

Wywołanie:

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

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

# Strona wykładu

http://bioexploratorium.pl/wiki/Wstęp_do_programowania_
_2014z