Komunikacja człowiek – komputer

Tutorial 3:

Podstawy języka Python

Opracował:
dr inż. Wojciech Bieniecki
wbieniec@kis.p.lodz.pl
http://wbieniec.kis.p.lodz.pl
Instytut Informatyki Stosowanej
Politechnika Łódzka

Czym jest Python

Python to popularny (gdyż prosty, o dużych możliwościach, a zarazem darmowy) język skryptowy, opracowany we wczesnych latach 90. przez Guido van Rossuma

Język nie wymusza jednego stylu programowania, pozwalając na stosowanie różnych. W Pythonie możliwe jest programowanie obiektowe, programowanie strukturalne i programowanie funkcyjne. Typy sprawdzane są dynamicznie, a do zarządzania pamięcią stosuje się garbage collection.

Zastosowanie Pythona

NASA – aplikacje do zarządzania kontrolą startową wahadłowców.
Projekt Nebula to rozproszone środowisko obliczeniowe przystosowane do wykonywania obliczeń w "chmurze" łączące wiele modułów i aplikacji Pythonowych.

Gdzie działa Python

YouTube - popularny serwis z klipami wideo jest w większości napisany w Pythonie. Twórcy serwisu podkreślali wydajność jaką oferuje Python, a także szybkie implementowanie nowych funkcjonalności poprzez czytelny kod, który łatwo rozszerzać i aktualizować.

Google używa Pythona w wielu swoich aplikacjach i usługach takich jak Google App Engine, czy Google Wave. Zatrudnia nawet twórcę tego języka - Guido van Rossuma.

Aplikacje napisane w Pythonie działają pod wieloma systemami takimi jak Windows, Linux/Unix, Mac OS X, OS/2, Amiga, czy smartphony Palma i Nokia. Dostępne są także implementacje Pythona w Javie (Jython) i .NET (IronPython) działające wszędzie tam, gdzie dostępne są te platformy.

Typy projektów

Tworzenie dynamicznych stron internetowych
Frameworki **Django**, **Pylons**, serwer aplikacji **Zope/Plone**.
Efektywnye i szybkie tworzeniu nowoczesnych stron internetowych bogatych w funkcjonalności.
Język ma wiele zalet w porównaniu do PHP

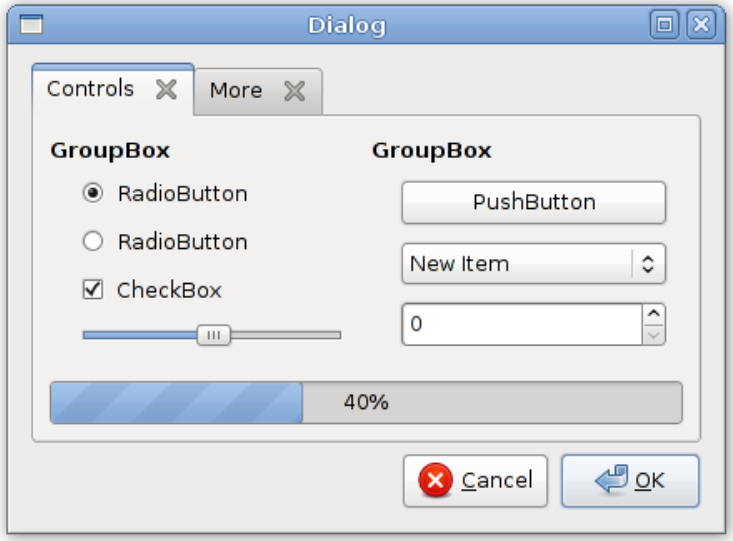
Platforma **Google App Engine** dla rozproszonego hostingu aplikacji internetowych. Oparta została o Pythona i oferuje serwisom www taką samą skalowalność, jaką posiadają wszystkie aplikacje i usługi tej firmy. GAE jest darmowe.

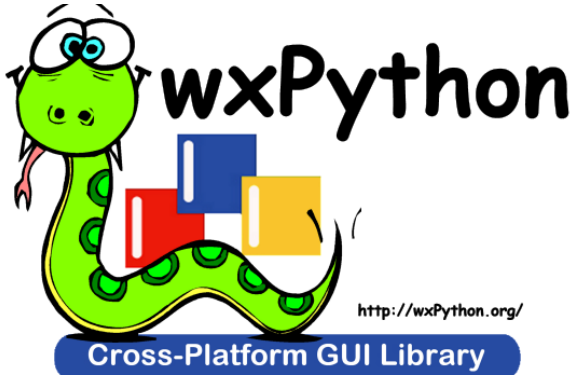
Usługi i serwisy społecznościowe to obecnie podstawa dla wielu serwisów www.
Za pomocą Pythona można wykorzystać API/usługi serwisów takich jak Twitter, Blip.pl, Facebook, aplikacji Google (Maps, Docs i innych przez GData), czy Google Wave.
Przykładowo biblioteka *PyFacebook* pozwala tworzyć aplikacje dla tego serwisu , a API Google Wave umożliwia tworzenie aplikacji dla tej platformy, czy integrowanie własnych stron i aplikacji z Wave

Typy projektów

Programowanie sieciowe - różne usługi, biblioteki, aplikacje, serwery i klienty wykorzystujące sieci. Moduły: socket, select, HTTPServer, SimpleHTTPRequestHandler

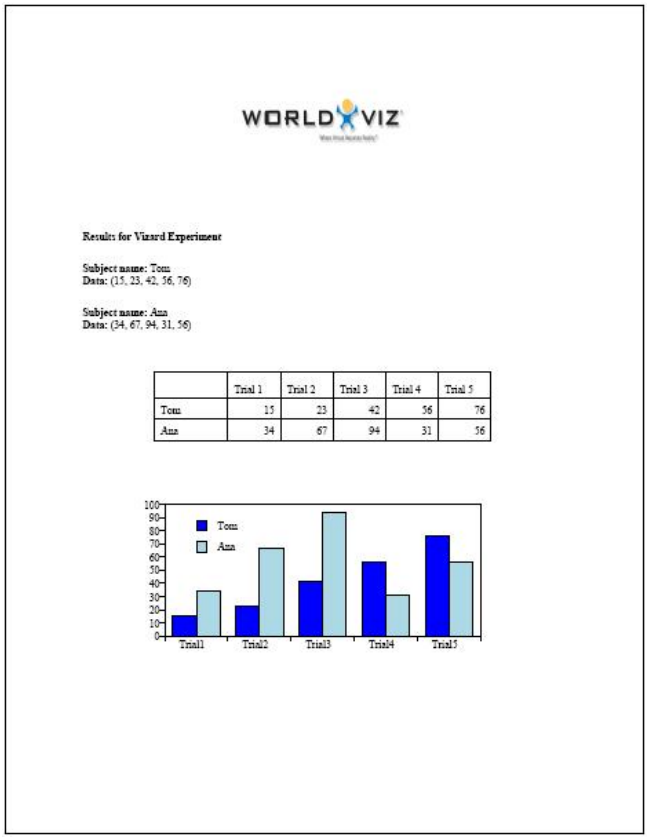
Aplikacje desktopowe (MS Windows, OS X, Linux) można pisać w Pythonie za pomocą bibliotek takich jak PyQt4, PyGTK, wxPython, czy wbudowanej biblioteki tk. Za pomocą aplikacji py2exe można stworzyć gotowe aplikacje (exe) dla systemów MS Windows, a za pomocą py2app gotowe aplikacje dla OS X.



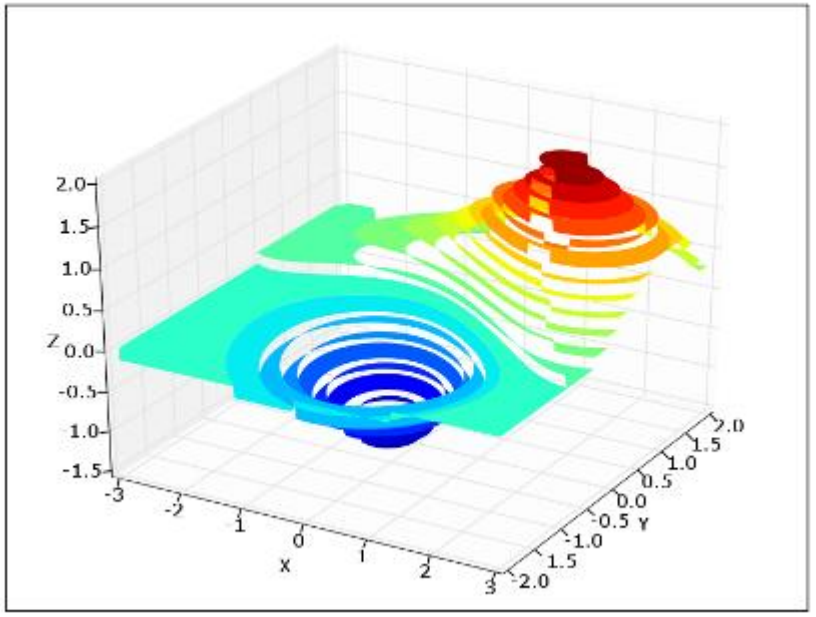


Typy projektów

Zastosowania naukowe, finansowe, przetwarzanie danych.
Generowanie wykresów, zestawień, plików PDF, arkuszy Excela, czy ODT/ODS.
Reportlab generuje pliki PDF i formatuje raporty



Matplotlib - generowanie wykresów.



Typy projektów

Zastosowania naukowe, finansowe, przetwarzanie danych.
Scipy oferuje implementacje różnych algorytmów do metod numerycznych.
PIL – biblioteka przetwarzania i analizy obrazów
xlwt i **xlrt** zapis i odczyt arkuszy Excela.

Gry i aplikacje wykorzystujące 3D też można stworzyć z wykorzystaniem Pythona.

Bazy danych możliwość zarządzania bazami danych różnych typów, głównie ORACLE.

Python 2 i Python 3

Python 3 = Python 3000 lub Py3K

Wyrażenie **print** zastąpiono przez funkcję **print()**

Metody dict.keys(), dict.items() i dict.values() zwracają **widoki** zamiast **list**

map() i filter() zwracają iteratory

Użycie typów text i binarnego zamiast stringów unikodowych i 8-bitowych

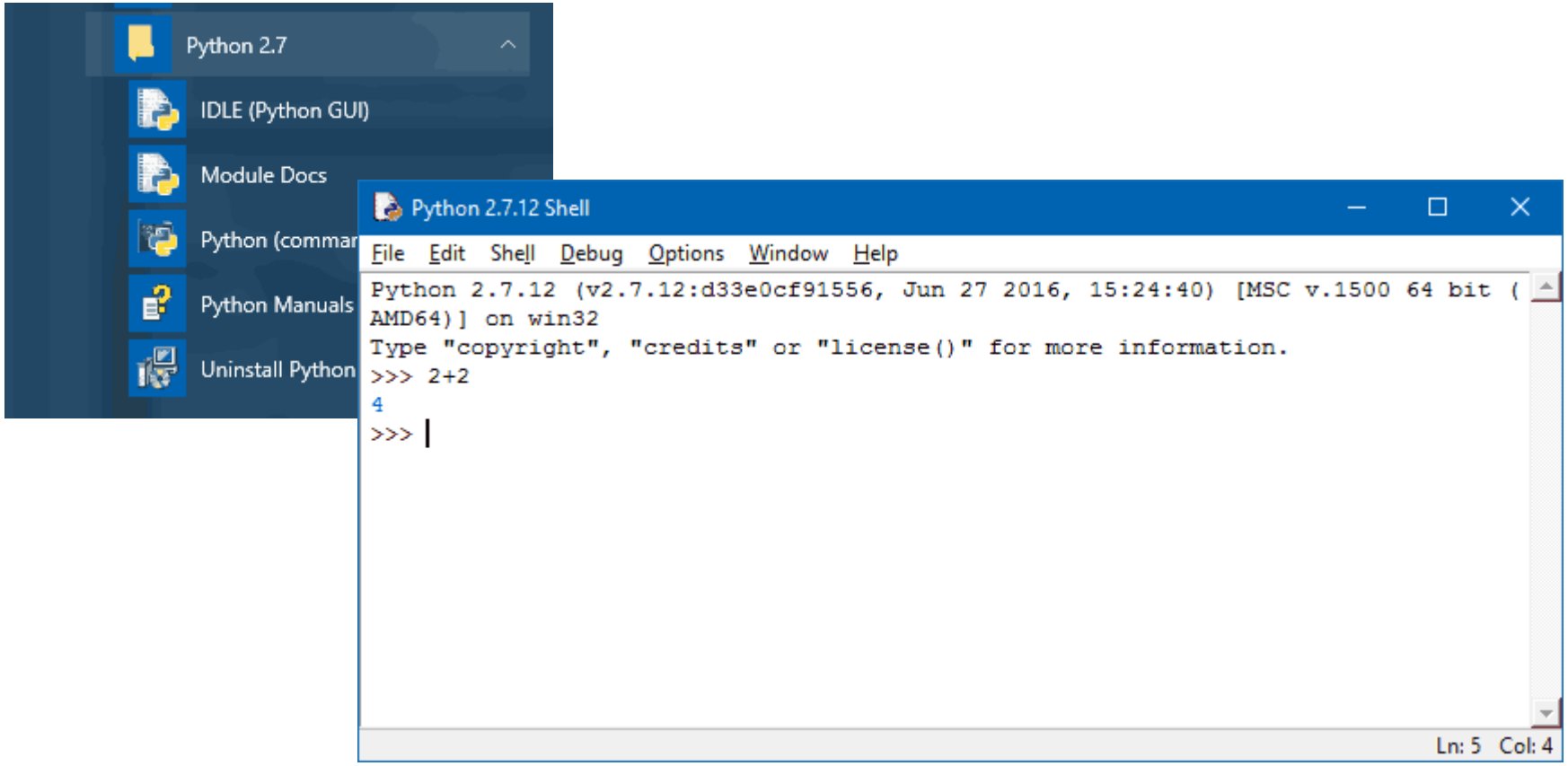
Niezgodność wielu przydatnych bibliotek

Instalacje http://www.python.org/getit/

Python 2.7.12 i Python 3.5.2.

Wersje 32 bitowe i 64 bitowe na Windows oraz na inne systemy: Linux, MacOS

Uruchamianie środowiska tryb interaktywny (REPL)



Oczywiście, możliwe jest uruchamianie plików w trybie wsadowym (pliki *.py) a także kompilowanie plików. Kompilacja do kodu bajtowego (do pliku .pyc) następuje przy pierwszym uruchomieniu skryptu.

Podstawowe operacje

Przypisanie
liczba = 2
urzad = 'pocztowy'
zespolona = 1 + 1j

Przypisanie wartości więcej niż jednej zmiennej
a = b =c = 10

Przypisanie wartości innej zmiennej
kopia_liczby = liczba, znaczek = urzad

Zamiana wartości miejscami
znaczek, liczba = liczba, znaczek

Usunięcie zmiennej
del kopia_liczby; del urzad #czy znaczek nadal istnieje?

Przetwarzanie napisów

```
print "A\nZ"

print 'A\nZ'

print """A\nZ"""
```

Napisy surowe

```
print r"A\nZ"

print r'A\nZ'

print r"""A\nZ"""
```

```
napis = 'ala ma kota.'  # napis jest obiektem
```

Powiększenie pierwszej litery

```
napis.capitalize() # 'Ala ma kota.'
```

Zliczenie wystąpień znaków

```
napis.count('a') # 4
```

Wycentrowanie tekstu (ljust, rjust)

```
napis.center(20) # ' Ala ma kota. '
```

Kodowanie napisu (decode)

```
napis.encode('iso8859-2') # 'ala ma kota'
```

Przetwarzanie napisów

Sprawdzenie czy jest zakończony na (startswith)
napis.endswith('kota.')

Zamiana tabulatorów na spacje
'tab\ttab'.expandtabs(1)

Znalezienie miejsca pierwszego wystąpienia (index, rfind, rindex)
napis.find('ma')

Sprawdzenie czy alfanumeryczny (isalnum, isdigit, islower, isupper, isspace, istitle; unicode: isnumeric, isdecimal)
'77A'.isalnum()

Wycinanie znaków z początku i końca (lstrip, rstrip)
napis.strip('a.')

Dzielenie stringu na dwie części (split, splitlines, rsplit, rpartition)
napis.partition('ma')

Przetwarzanie stringów

Podmiana fragmentu
napis.replace('ala', 'Ola').replace('.', ' i psa.')

Zmiana na tytuł (lower, upper, swapcase)
napis.title() # 'Ala Ma Kota.'

Konkatencja stringów
'Ala' "Ma" """Kota""" # 'AlaMaKota'

Konwersja na string
str(cokolwiek)

Operator konkatencji
'Ala ma ' + str(2) + """ koty.""" # 'Ala ma 2 koty'

Operator powielania
'la'*7 # 'lalalalalalala'

Przetwarzanie stringów

Podawanie kolejnych parametrów:
"Podajemy {0} paramet{1}".format('kolejne', 'ry')

Parametry nazwane
"Nazwa {param}".format(param='parametru.')

Pojedynczy znak (liczymy od 0)
napis[4]
Zakres (liczymy od 0)
napis[4:6]
Pojedynczy znak od końca
napis[-5]
Zakres licząc od końca
napis[-5:-1]
napis[-5:]

Zagadka: Jak działa ta operacja?
napis[::-1]
napis[::-2]

UWAGA!
String jest obiektem niemutowalnym.
Przy modyfikacji stringu zwracana jest kopia obiektu.
Oryginalna zmienna nie ulega modyfikacji !

Praca z listą

Tworzenie pustej listy
lista = []

Modyfikacja list
lista = ['a', 'b', 'c']

Dodawanie nowych elementów
lista.append('d')

Rozszerzanie listy o listę nowych elementów
lista.extend(['e', 'f', 'g'])

Dodanie elementu na zdefiniowanej pozycji
lista.insert(0, '0')

Praca z listą

Usunięcie pierwszego wystąpienia elementu
lista.remove('0')

Usunięcie elementu z pozycji
lista.pop (3)

Usunięcie elementu z pozycji
lista.del (3)

Odwrócenie kolejności
lista.reverse()

Sortowanie listy
lista.sort()

Pobranie indeksu zadanego elementu
lista.index('d')

Liczba elementów listy
len(lista)

Liczba konkretnych elementów listy
lista.count('c')

Praca ze słownikiem

Tworzenie pustego słownika
di={}

Inicjalizacja nowego słownika
di={'a': 'A', 'b': 'B', 'c': 'C'}

Aktualizacja słownika
di.update({'d':'D'})

Wyświetlenie listy kluczy (iterkeys)
di.keys()

Wyświetlenie listy wartości (itervalues)
di.values()

Wyświelenie listy elementów (iteritems)
di.items()

Usunięcie wartości o zadanym kluczu
di.pop('d')

Usunięcie pierwszego elementu
di.popitem()

Wyczyszczenie słownika
di.clear()

Praca z krotkami

Krotka (ang. *tuple*) jest niezmienną listą. Zawartość krotki określamy tylko podczas jej tworzenia. Potem nie możemy już jej zmienić.

Tworzenie nowej krotki

```
t = ("a", "b", "ala", "z", "element")
```

Wybór elementów krotki

```
t[0]
t[-1]
t[1:3]
```

Przeszukiwanie krotki

```
t.index("z")
"z" in t
```

Krotki można konwertować na listy.
*Wbudowana funkcja **tuple**, której argumentem jest lista, zwraca krotkę z takimi samymi elementami, natomiast funkcja **list**, której argumentem jest krotka, zwraca listę.*
*W rezultacie **tuple** zamraża listę, a **list** odmraża krotkę.*

Praca ze zbiorami

Zbiory to nieuporządkowane zestawy obiektów. Używamy ich, gdy istotny jest tylko fakt występowania elementu, a nie jego położenie albo liczba powtórzeń.

Zbiory można testować pod kątem występowania danego elementu, sprawdzać czy to jest podzbiór innego zbioru, szukać części wspólnej zbiorów.

Tworzenie zbioru

```
A = set()
kraje = set(['Brazylia', 'Rosja', 'Indie'])
```

Modyfikacja zbioru

```
kraje2 = kraje.copy()
kraje2.add('Chiny' )
```

... lub szybciej ...

```
kraje2 = kraje | set(['Chiny'])
kraje.remove('Rosja')
```

Praca ze zbiorami

Czy element występuje w zbiorze?

```
>>> 'Indie' in kraje
True
>>> 'USA' in kraje
 False
```

Zawieranie się zbiorów

```
kraje2.issuperset(kraje)
kraje2 > kraje
kraje.issubset(kraje2)
kraje < kraje2
```

Część wspólna zbiorów

```
kraje.intersection(kraje2)
kraje & kraje2
```

Pętle w Pythonie

Filozofia Pythona: jeśli możesz – unikaj pętli

Pętle w Pythonie tworzymy wykorzystując sekwencje

range(10)

range(1,10)

range(1,10,2)

range(9,0,-1)

```
for x in range(3,10):
    print x,'** 2 =',x*x
```

Pętle zagnieżdżone

Przykład - Wygeneruj tabliczkę mnożenia

```
for x in range(1,11):
    print # przejście do nowego wiersza
    for y in range(1,11):
        print "%3i" % (x*y),
```

Instrukcje break i continue

```
liczby = input("Podaj kilka liczb oddzielając przecinkiem:")
for x in liczby:
    if x<0: continue
    print "Pierwiastkiem liczby %2i jest %5.3f" % (x,x**0.5)
```

```
liczby=[1,3,4,5,7,8,9,11]
szukana=5
for p,x in enumerate(liczby):
    if x != szukana: continue
    print "Znaleziono liczbę %i na pozycji %i" % (x,p+1)
    break;
```

Pętla while

Przykład: co wyświetli ten program?

```
a=[1,2,3,4,5,6]
while a:
    a=a[:len(a)-1]
    print a
```

Pętla typu **while** może również zawierać blok po **else**, wykonywany po ostatnim obiegu pętli:

```
a=7
while a:
    a-=1
print a
else:
    print "koniec"
```

Szybkie przetwarzanie list

W Pythonie w większości przypadków nie ma konieczności używania pętli w czasie przetwarzania list

Szybkie tworzenie listy

```
Lista = [0] * 20

Lista2 = [3,5,2] * 6

Lista3 = range(1,21,2)
```

List comprehensions (pol. wytworniki list).

Jest to wygodne narzędzie do szybkiego tworzenia zaawansowanych list oparciu o inną listę.

List comprehensions

Postać prosta [*wyrażenie* for *zmienna* in *sekwencja*]

```
Lista4 = [2*x for x in Lista3]

Lista_krotek = [(x, x*x) for x in range(1,5)]

Kody_ASCII = [(x, ord(x)) for x in "ABCDEFGHIJKLMNOPQRSTUVWXYZ"]
```

Postać prosta warunkowa
[*wyrażenie* for *zmienna* in *sekwencja* if *warunek*]

```
Lista5 = [x for x in Lista3 if x>10]
Podzielne = [x for x in range(1,20) if not (x%3) or not (x%5)]
Samogloski = [(x, ord(x)) for x in "ABCDEFGHIJKLMNOPQRSTUVWXYZ" if x in "AEIOUY"]
```

List comprehensions

Postać rozszerzona

[*wyrażenie* for *zmienna1* in *sekwencja1* for *zmienna2* in *sekwencja2* ...]

Iloczyn kartezjański sekwencji

```
Pary = [(x,y) for x in range(1,5) for y in range(4,0,-1)]
```

```
Roznice = [x-y for x in range(1,5) for y in range(4,0,-1)]
```

```
Sklejka = [str(x)+y+str(z) for x in [1,2] for y in ['A','B'] for z in [0,3] ]
```

Postać rozszerzona z jednym warunkiem

[*wyr* for *zm1* in *sekwencja1* for *zm2* in *sekwencja2* ... if *warunek*]

Pary każdy element z każdym, tylko jeżeli pierwszy element jest mniejszy od drugiego:

```
[(x,y) for x in range(1,5) for y in range (6,3,-1) if x<y]
```

Postać rozszerzona z wieloma warunkami

[wyrażenie for zmienna1 in sekwencja1 if warunek1
for zmienna2 in sekwencja2 if warunek2 ...]

Wykonywanie funkcji na listach

Wyrażenie lambda służy do definiowania *anonimowych funkcji* w postaci wyrażenia:
lambda argument : wyrażenie

```
a = lambda : 'a'
print a()
b = lambda x,y : x+y
print b(1,2)
```

apply wywołanie funkcji z parametrami uzyskanymi z rozpakowania sekwencji

```
dziel=lambda x,y,z: (x+y)/z
#zamiast pisać dziel(3,5,2)
xyz=(3,5,2)
apply(dziel,xyz)
```

Wykonywanie funkcji na listach

Funkcja **map()** wykonuje podaną funkcję dla każdego elementu listy, krotki lub słownika.
Często łączy się z funkcją anonimową.
Wynikiem operacji jest lista

```
map(lambda x: x*x, range(5))

map(dziel, range(5), range(5), [2]*5)
```

Map często jest używane do konwersji listy

```
lista_str = map(str, range(1, 11))

print map(len, "Ala ma kota".split())
```

Wykonywanie funkcji na listach

Funkcja **zip()** służy do **konsolidacji danych**.
Przyjmuje jako swoje parametry jedną lub więcej sekwencji, po czym zwraca listę krotek, których poszczególne elementy pochodzą z poszczególnych sekwencji.

```
Krotki = zip("abcdef",[1,2,3,4,5,6])
```

gdy długości sekwencji są różne, wynikowa sekwencja jest skracana do najkrótszej

Przykład: Jaka będzie różnica w działaniu map i zip poniższym kodzie?

```
S1 = 'abc'
S2 = 'xyz123'
print zip(S1, S2)
print map(None, S1, S2)
```

Wykonywanie funkcji na listach

Funkcja **filter()** filtruje elementy sekwencji przy użyciu podanej funkcji, która musi zwracać wartość Prawdę lub Fałsz

```
S="Ala ma kota i konto na chomiku."
Samogloski=filter(lambda x:x in 'aeiouy' , S)
Pozostale = filter(lambda x:x not in in 'aeiouy' , S)
```

Z ciągu liczb naturalnych <2, 24> wybierz takie, które nie są podzielne przez 2 ani 3

```
filter(lambda x: x % 2 != 0 and x % 3 != 0, range(2, 25))
```

Wykonywanie funkcji na listach

Funkcja **reduce()** pobiera dane z sekwencji i zwraca na ich podstawie pojedynczą wartość np. sumę liczb.
Funkcja ta wykonuje podaną jako pierwszy argument funkcję dla pierwszych dwóch elementów sekwencji, a następnie wykonuje tą funkcję dla wyniku i trzeciego elementu - i tak do wyczerpania elementów sekwencji.

```
#suma elementów:
reduce(lambda x,y: x+y, [5,2,3,6])
#prościej
sum([5, 2, 3, 6])

#iloczyn elementów (np. silnia):
reduce(lambda x,y: x*y, [1,2,3,4])


#suma kwadratów elementów:
 reduce(lambda x,y: x+y, map(lambda x: x*x, range(1,10)))
```

Zagadka: spróbuj zamienić przykłady z map, zip, filter, reduce na *list comprehensions*

Sortowanie list

```
l=[3,2,5,7]
l.sort()
l.sort(reverse=True)
```

Przykład: posortuj wyrazy alfabetycznie

```
s="Ala i Ola mają kota oraz psa"
L=s.split()
L.sort()
L.sort(key=str.lower)
```

Przykład: posortuj ciągi znaków zawierające liczby

```
L=["11","2","20","7","55"]
L.sort()
L.sort(key=int)
```

Przykład: Tworzenie własnej funkcji sortującej

```
L=[ ("Adam",15), ("Bogdan",19), ("Ala",17), ("Zenobia",
14) ]
L.sort()
```

Przykłady praktyczne

1. Poproś użytkownika o podanie dwóch napisów, przy czym drugi z nich musi być 1-literowy (jeśli nie jest, wypisz komunikat o błędzie). Następnie napisz, ile razy drugi napis mieści się w pierwszym.

2. Poproś użytkownika o podanie napisu s o długości przynajmniej 10, a nie więcej niż 20 (w razie potrzeby wypisz komunikat o błędzie), a następnie utwórz string będący 10-krotnym powieleniem litery środkowej s (czyli np. przy długości 13 będzie nią s[6]).

3. Policz 15!, wypisz na ekranie tylko 3 pierwsze cyfry tej liczby.

4. Znajdź i wypisz na ekranie wszystkie liczby pierwsze z przedziału [20, 100].

Przykłady praktyczne

5 Wygeneruj na ekranie tabliczkę mnożenia 10 x 10

6 Wczytaj liczbę całkowitą (być może ujemną), a następnie wypisz jej cyfry słownie. Wykorzystaj słownik.

7 Ile jest różnych znaków w treści tego zadania? Użyj zbioru (set).

8 Wczytać napis złożony z co najmniej 3 słów (np. jakieś zdanie), a następnie wypisać string będący akronimem, czyli sklejeniem pierwszych liter tych słów, zamienionych na wielkie litery. Przykład: "Fatalna imitacja auta turystycznego, 1-miejscowa, 2-cylindrowa, 6-krotnie przeplacona" -> FIAT126P.