Spis treści

Python – instalacja

Instalacja interpretera

W przypadku Linux'a Python jest standardowo zainstalowany w systemie. W podstawowej wersji nie trzeba go dodatkowo konfigurować.

W przypadku Windowsa Pythona należy zainstalować. Należy pobrać interesującą nas wersję z https://www.python.org/download/ . Po zainstalowaniu trzeba upewnić się, że katalog Pythona, oraz katalog Script w katalogu Pythona są dodane do zmiennej systemowej PATH. Jeśli nie, to należy je dodać ręcznie, a następnie zrestartować komputer.

Instalacja dodatkowych bibliotek (PyPi)

Często zdarza się, że potrzebujemy dodatkowych bibliotek. Używając Pythona, nie ma z tym problemu, o ile znajdziemy taką, która nas interesuje. Dobrą bazą bibliotek jest https://pypi.python.org/pypi . Oczywiście poza nią możemy szukać na własną rękę np. przy użyciu Google.

Po ściągnięciu gotowego pakietu należy go zainstalować. Aby to zrobić musimy rozpakować pakiet, a następnie z linii poleceń przejść do katalogu, w którym znajduje się plik setup.py.Następnie musimy po kolei wykonać polecenia:

```
setup.py build
setup.py install
```

Od tej chwili możemy w naszym programie importować pakiet i go używać.

Alternatywą jest zainstalowanie programu do zarządzania pakietami (PIP). Instalujemy go analogicznie do zwykłego pakietu. Ma on kilka przydatnych opcji jak instalowanie, szukanie, odinstalowywanie pakietów, które pozwalają nam na szybkie zarządzanie pakietami bez fizycznego szukania w internecie. Najczęściej używane polecenia:

```
pip install nazwa_pakietu
pip search nazwa_pakietu
pip uninstall nazwa_pakietu
```

Konfiguracja edytora

Programowa w Pythonie możemy w dowolnym edytorze. Na początek polecam coś prostego typu Notepad++. Na początku najważniejsze jest, żeby skonfigurować 2 rzeczy: zmianę tabulatora na spację (najlepiej 4) oraz ustawienie kodowania każdego pliku na UTF-8 (najlepiej bez BOM).

Python – podstawy języka

Ogólne cechy charakteryzujące język

Ogólne cechy charakteryzujące język:

- brak nawiasów definiujących bloki – odpowiednie wcięcia
- brak średnika na końcu instrukcji
- rozszerzenie pliku *.py

Kodowanie tekstu

Aby polskie znaki poprawnie się wyświetlały w konsoli, na początku pliku należy wpisać:

```
#-*- coding: utf-8 -*-
```

Plik oczywiście musi być zapisany w formacie UTF-8 (bez BOM).

Komentarze

Komentarz jednoliniowy zaczyna się od znaku #. Przykłady:

```
#komentarz jednoliniowy
print 'testowy napis' #komentarz
```

Zmienne

Zmienne w Pythonie nie wymagają określenia typu.

```
zmienna = 'tresc zmiennej' #zadeklarowanie i ustalenie wartości zmiennej
zmienna  = 15 #zmiana zawartości zmiennej
zmienna2 = False
```

Występuje tu tak zwane kacze typowanie: Jeśli coś wygląda i kwacze jak kaczka, to na pewno jest kaczką.

Wyświetlanie tekstu

Do wyświetlania tekstu służy polecenie print. Przykład:

```
print 10
print 'napis'
print u'napis z polskimi znakami, np. óźł'
imie = 'Jarek'
print imie
print u'Mam na imię %s.' % imie
print u'Mam na imię %s, mam %d lat.' % (imie, 15)
print u'imię:'+imie
```

Operatory logiczne

W Pythonie występują takie same operatory logiczne, jak w innych językach: <, <=, >, >=, ==, !=, not, and, or.

```
a = 15
c = 0
print a < c #wyświetli False
print bool(a) and bool(c) #wyświetli False
print bool(a) or bool(c) #wyświetli True
```

Operatory arytmetyczne

Występują tu podobne operatory jak w innych językach: +, -, *, / (dzielenie), % (reszta z dzielenia), ** (potęgowanie), // (dzielenie całkowite), +=, itd.

Brak tutaj niestety operatorów inkrementacji i dekrementacji (–, ++).

```
print 15/2 #wynikiem będzie 7
print float(15)/2 #wynikiem będzie 7.5 - rzutowanie
print 15.0/2 #wynikiem będzie 7.5
a = 30
a += 15 #a = a + 15
print a #wynikiem będzie 45
```

Przy okazji działania na liczbach poznamy jeszcze proste funkcje wbudowane:

```
#zaokrąglenie liczby do miejsca po przecinku:
print round(2.8775, 2) #wyświetli 2.88
#moduł liczby (wartość bezwzględna):
print abs(-15.5) #wyświetli 15.5
```

Sekwencje

Operacje na sekwencjach na przykładzie string'a

Poniżej przykład operacji na sekwencjach na przykładzie string'a:

```
zmienna = 'napis'

#wykrojenia
print zmienna[0] #pierwszy znak ciągu, wyświetli "n"
print zmienna[-1] #ostatni znak ciągu (pierwszy od końca), wyświetli "s"
print zmienna[1:-1] #wycięcie fragmentu stringa, wyświetli "api"
print zmienna[1:] #wycięcie fragmentu stringa, wyświetli "apis"

#operator łączenia
print 'zmienna: '+zmienna #wyświetli "zmienna: napis"

#operator powtarzania
print '-'*5 #wyświetli "-----"

#operator in, not in
print 'api' in zmienna #wyświetli True
print 'api' not in zmienna #wyświetli False
```

String w Pythonie jest niemodyfikowalny. Znaczy to tyle, że zmieniając string tak naprawdę przypisujemy do zmiennej zupełnie nowy.

Listy

Listy są podobnie jak stringi są sekwencjami. Możemy zatem wykonywać na nich te same operacje.

Lista w Pythonie przypomina tablice z innych języków programowania. Nie ma ona z góry określonej wielkości, można do niej dodawać elementy oraz je usuwać (czyli modyfikować – w przeciwieństwie do stringów).

```
#utworzenie listy
lista = ['pierwszy', 'drugi', 'trzeci']
```

```
#dodanie elementu na końcu listy
lista.append('czwarty')
#wstanie elementu na wybranej pozycji
lista.insert(1, 'na indeksie 1, czyli 2 pozycja')
#usuń z listy element "drugi"
lista.remove('drugi')
#uzuń element o zadanym indeksie
lista.pop(-1)
#duplikowanie listy
lista_podwojna = lista*2
#rozszeżenie listy
lista.extend(['4', '5'])
#sortowanie listy
lista.sort()
```

Krotki

Krotki są podobne do list. Różnica jest taka, że są one niemodyfikowalne.

```
krotka = ('pierwszy', 'drugi')
print krotka[0] #wyświetli "pierwszy"
krotka = ('pierwszy',) #tworzy jednoelementową krotkę
krotka = ('pierwszy') #tworzy ciąg znaków - ważny jest przecinek na końcu
```

Słowniki

Słownik jest modyfikowalnym, nieuporządkowanym odwzorowaniem kluczy na wartości.

```
slownik = {'pierwszy':'1wszy', 'drugi':'2gi'}
print slownik.get('pierwszy', 'brak') #wyświetli "1wszy"
print slownik.get('trzeci', 'brak') #wyświetli "brak"
print slownik['pierwszy'] #wyświetli "1wszy"
#print slownik['trzeci'] #otrzymamy błąd - brak klucza
print slownik.setdefault('trzeci', '3ci') #pobierze wartość o kluczu
"trzeci", jeśli nie istanieje utworzy ją i doda dla klucza wartość "3ci"
del slownik['drugi'] #usunie klucz i wartość, gdzie klucz równy jest "drugi"
slownik['pierwszy'] = 'pierwszy!' #nadpisuje wartość dla podanego klucza
```

Instrukcje warunkowe

```
zmienna = 2
if zmienna == 13:
      print u'Tak! Zmienna ma wartość 13!'
elif zmienna == 0:
      print 'A nie bo 0.'
else:
      print u'Nie! Zmienna ma wartość %s!' % zmienna
```

Pętle

```
for x in range(1,10):
      print x

zmienna = [2, 5, 18]
for x in zmienna:
      print x**2
```

```
zmienna = [5, 9, 78, 2, 25, 32]
for x in zmienna:
      if x%2==0:
            print 'Liczba %d jest parzysta!' % x
      else:
            print 'Liczba %d jest nieparzysta!' % x
```

Funkcje

Podstawy

Funkcja służy do zebrania powtarzającego się kodu w jednym miejscu.

```
def funkcja_liniowa(a, b):
      x = float(-1*b) / a
      return x

print funkcja_liniowa(2, -3) #wyświetli 1.5
print funkcja_liniowa(8, -16) #wyświetli 2.0
```

Domknięcia

Domknięcie polega na zdefiniowaniu obiektu działającego w pewnym środowisku, np.:

```
def mnozenie_przez(mnoznik):
      def domnoz(mnozna):
            return mnozna * mnoznik
      return domnoz

iloczyn_5_przez = mnozenie_przez(5)
print iloczyn_5_przez(12)     #wyświetli 60
```

Wyrażenia lambda

Wyrażenia lambda działają podobnie jak funkcje anonimowe w innych językach.

```
#funkcja
def f(x):
      return x*2
print f(3)
#przypisanie wyrażenia lambda do zmiennej
l = lambda x: x*2
print l(3)
#bezpośrednie użycie wyrażenia lambda
print (lambda x: x*2)(3)
```

Iterator i generator

Iterator jest obiektem, który pozwala na sekwencyjny dostęp do wszystkich elementów lub części zawartych w obiekcie. Python zawiera wbudowane mechanizmy iteracji, gdzie iterator nie jest wprowadzany jawnie:

```
for x in range(1,10):
      print x
```

Generator jest obiektem, który pozwala na sekwencyjny dostęp do obiektu – nie tworząc go od razu w całości:

```
generator = (x for x in range(1,10))
print generator #wyświetli <generator object <genexpr> at 0x0000000001CAF168>
print [x**2 for x in generator if x <= 4] #wyświetli [1, 4, 9, 16]
```

Wyjątki

Wyjątki pozwalają na przejęcie obsługi błędu. Dzięki nim możemy zabezpieczyć program przed niepożądanym zatrzymaniem wykonania.

```
# zmienna = 1
try:
      print u'Wartość zmiannej: %s' % zmienna
except Exception, e:
      print 'Brak zmiennej: ', e
finally:
      print u'Ten kod zawsze będzie wykonany!'
```

Klasy i obiekty

Podstawy

W Pythonie większość elementów jest tak naprawdę obiektem. Jak każdy obiektowy język programowania, pozwala na tworzenie klas, a na ich podstawie obiekty.

```
class Dzialania:
      a = 0
      b = 0

      def __init__(self, x, y):
            self.a = x
            self.b = y

      def dodaj(self):
            return self.a + self.b

      def odejmij(self):
            return self.a - self.b

obiekt = Dzialania(1,5)
obiekt2 = Dzialania(8,4)
print obiekt.dodaj() #wyświetli 6
print obiekt.odejmij() #wyświetli -4
print obiekt2.odejmij() #wyświetli 4
```

Obiekt wywoływalny

Możemy łatwo sprawić, aby obiekt oparty na klasie był wywoływalny. Wystarczy dopisać metodę __call__:

```
class Dzialania:
      a = 0
      b = 0

      def __init__(self, x, y):
            self.a = x
            self.b = y
```

```
	def __call__(self):
		return 'Wprowadzone liczby: %s i %s.' % (self.a, self.b)

	def dodaj(self):
		return self.a + self.b

	def odejmij(self):
		return self.a - self.b

obiekt = Dzialania(1,5)
print obiekt() #wyświetli napis "Wprowadzone liczby: 1 i 5."
```

Wielodziedziczenie

Python pozwala na wielodziedziczenie, czyli dziedziczenie po wielu klasach. Jeśli w klasach po których dziedziczy klasa występują te same metody, to brana jest pod uwagę ta, która pierwsza została wywołana (w przykładzie Dodawanie). Można jednak wewnątrz klasy dziedziczącej wywołać metodę z klasy dziedziczonej, która nas interesuje i udostępnić pod inną metodą.

```
class Dodawanie:
	def dodaj(self):
		return self.a + self.b

class Odejmowanie:
	def odejmij(self):
		return self.a - self.b

class Dzialania(Dodawanie, Odejmowanie):
	a = 0
	b = 0

	def dodawanie(self):
		return Dodawanie.dodaj(self)

	def __init__(self, x, y):
		self.a = x
		self.b = y

	def __call__(self):
		return 'Wprowadzone liczby: %s i %s.' % (self.a, self.b)

obiekt = Dzialania(1,5)
print obiekt.dodaj() #wyświetli 6
print obiekt.dodawanie() #wyświetli 6
print obiekt.odejmij() #wyświetli -4
print obiekt() #wyświetli napis "Wprowadzone liczby: 1 i 5."
```

Przeciążanie operatorów

Klasy w Pythonie pozwalają na przeciążanie operatorów.

```
class Procent:
	liczba = 0

	def __init__(self, wartosc):
		self.liczba = wartosc

	def __repr__(self):
		return '%d%%' % self.liczba
```

```
        def __add__(a, b):
              return '%d%%' % (a.liczba+b.liczba)

        def __mul__(a, b):
              return '%s%%' % (float(a.liczba)*b.liczba/100)

        def __float__(self):
              return float(self.liczba)/100

l1 = Procent(5)
l2 = Procent(85)
print l1 #wyświetli 5%
print l2 #wyświetli 85%
print l1+l2 #wyświetli 90%
print l1*l2 #wyświetli 4.25%
print float(l1)*200 #wyświetli 10.0
print '%d' % (float(Procent(50))*1000) #wyświetli 500
```

Moduły, pakiety, przestrzenie nazw

Moduł, to biblioteka funkcji napisanych przez nas, które znajdują się w oddzielnym pliku.

Plik modul.py:

```
#-*- coding: utf-8 -*-

def dodawanie(a,b):
      return a+b

def odejmowanie(a,b):
      return a-b
```

Właściwy plik:

```
#-*- coding: utf-8 -*-

import modul as dzialania #import modułu i nadanie mu nowej nazwy

print dzialania.dodawanie(1,5) #wyświetli 6
print dzialania.odejmowanie(100,86) #wyświetli 14
```

Jak widać w drugim pliku zaimportowaliśmy nasz moduł, oraz daliśmy mu nową nazwę (co nie jest konieczne).

Zmieńmy teraz sytuację. Plik modul.py został umieszczony w katalogu pakiet. W nim również został umieszczony pusty plik o nazwie __init__.py. Katalog taki nazywany jest pakietem. Wywołanie pakietu wygląda następująco:

```
#-*- coding: utf-8 -*-

import pakiet.modul

print pakiet.modul.dodawanie(1,5) #wyświetli 6
print pakiet.modul.odejmowanie(100,86) #wyświetli 14
```

To co widzimy przy wywołaniu („kropkowane nazwy modułów”) to właśnie przestrzeń nazw, pod którą dostępna jest funkcja.

Operacje na plikach

Na plikach możemy wykonywać operacje odczytu i zapisu. Należy pamiętać o zamknięciu pliku po użyciu. W celu zabezpieczenia się przed problemami z operacjami na plikach, należy używać składni try.

```
#-*- coding: utf-8 -*-

# otwarcie pliku tekstowego
plik = open('text.txt')
# otwarcie pliku binarnego
# text = open('plik', 'rb').read()
try:
    # operacje na plikach
    # odczytanie pliku i przypisanie zawartości do zmiennej
    text = plik.read()
finally:
    # zamknięcie pliku
    plik.close()

# wyświetlenie zawartości pliku
print text

# otwarcie pliku do zapisu
plik = open('plik', 'w') # plik tekstowy
# plik = open('plik', 'wb') # plik binarny
try:
    # zapis zmiennej tekstowej
    plik.write(text)
    plik.write('test')
    lista = ["bla ", "bla ", "yyy "]
    # zapis całej listy - separatorem jest spacja
    plik.writelines(lista)
finally:
    plik.close()

# zmiana fragmentu pliku
zrodlo = open('text.txt').readlines()
cel = open('text.txt', 'w')
for s in zrodlo:
      cel.write(s.replace("co zamienić", "na co"))
cel.close()

# pobieranie określonego wiersza pliku
import linecache
wiersz = linecache.getline('text.txt', 2)
print 'wiersz: %s' % wiersz

# usuwanie pliku jeśli istnieje
if os.path.isfile('text.txt') :
    os.unlink('text.txt')
```

Operacje na katalogach

```
#-*- coding: utf-8 -*-

import os
import shutil
```

```
# pobranie zawartości katalogu
print os.listdir('.')

for file in os.listdir('.\\instalatory'):
    print file

if not os.path.isdir('folder'):
    # tworzenie katalogu
    os.makedirs('folder')
    print u'folder został utworzony'
else:
    print u'folder już istnieje';
    # usuwanie katalogu - nawet nie pustego
    shutil.rmtree('folder')
```

Obsługa opcji podanych do skryptu

```
#-*- coding: utf-8 -*-

import sys

#wyświetlenie listy parametrów - pierwszy jest nazwą skryptu
print sys.argv[1:]

for arg in sys.argv:
    print arg
```

Uruchamianie zewnętrznych programów

Aby uruchomić z Pythonie zewnętrzny program, wystarczy napisać krótki skrypt:

```
import subprocess as sub

p = sub.Popen("C:\Program Files (x86)\Google\Chrome\Application\chrome.exe")
```

Skrypt powyżej uruchamia przeglądarkę Chrome na moim komputerze. W odpowiednim miejscu należy wstawić ścieżkę do programu, który chcemy uruchomić.

Możemy także bardzie zintegrować nasz skrypt z programem. Załóżmy, że chcemy „rozmawiać” z programem napisanym w C++, którego kod wygląda następująco:

```
#include
using namespace std;

int main()
{
    int in;
    cin >> in;
    cout < < "Wpisano liczbe: " << in << endl;
    return 0;
}
```

Uruchomienie takiego programu i przejęcie kontroli nad jego wejście/wyjściem wygląda następująco:

```
#-*- coding: utf-8 -*-

import subprocess as sub

p = sub.Popen(["program.exe"], stdin=sub.PIPE, stdout=sub.PIPE,
stderr=sub.STDOUT)
```

```
child_output, child_error = p.communicate(input="234")
print child_output
```

Komunikacja po TCP/IP

Serwer:

```
#-*- coding: utf-8 -*-

from socket import *
import time
s = socket(AF_INET, SOCK_STREAM) #utworzenie gniazda
s.bind(('', 8787)) #dowiazanie do portu 8888
s.listen(5)

while 1:
      client,addr = s.accept() # odebranie polaczenia
      print 'Polaczenie z ', addr
      tresc = client.recv(1024)
      client.send('Polaczenie z serwera, czas: %s, odebrana tresc: %s' %
(time.ctime(time.time()), tresc)) # wyslanie danych do klienta
      client.close()
```

Klient:

```
#-*- coding: utf-8 -*-

from socket import *
s = socket(AF_INET, SOCK_STREAM) #utworzenie gniazda
s.connect(('localhost', 8787)) # nawiazanie polaczenia
s.send('tresc z klienta!');
tm = s.recv(1024) #odbior danych (max 1024 bajtów)
s.close()
print 'Czas serwera: ', tm
```

Linki:

- http://www.python.rk.edu.pl/w/p/python-i-programowanie-sieciowe/
- http://www.tutorialspoint.com/python/python_networking.htm

Pobieranie strony HTML

```
#-*- coding: utf-8 -*-

import urllib
sock = urllib.urlopen('http://google.pl')
htmlSource = sock.read()
sock.close()
print htmlSource
```

Połączenie z FTP

Pobieranie plików:

```
#-*- coding: utf-8 -*-

import ftplib

# otwieramy połączenie
ftp = ftplib.FTP("ftp.serwer.jakiś")
ftp.login("login", "haslo")
```

```
# przejście do katalogu
ftp.cwd('public_html')

#pobranie pliku (binarnego)
f = open("nazwapliku", "wb")
ftp.retrbinary("RETR nazwapliku_na_serwerze", f.write)
f.close()

# zakończenie połączenia
ftp.quit()
By wysłać plik na serwer poprzez FTP:
import ftplib
```

Wysyłanie plików:

```
#-*- coding: utf-8 -*-

# otwieramy połączenie
ftp = ftplib.FTP("ftp.serwer")
ftp.login("login", "haslo")

# przejście do katalogu
ftp.cwd('public_html')

# otwarcie pliku
f = open("nazwa_pliku", "r")
ftp.storlines("STOR nazwa_pliku_jaka_ma_byc_na_serwerze", f)
f.close()

#binarny plik:
f = open("footprint.jpg", "rb")
ftp.storbinary("STOR footprint-1.jpg", f, 1024)
f.close()

# zakończenie połączenia
ftp.quit()
```

Tworzenie prostych programów okienkowych (EasyGui)

Przy użyciu biblioteki EasyGui (nie jest standardowo dostarczona z Python'em) możemy tworzyć proste programy oparte na okienkach. Przykładowy kod:

```
import easygui as eg
import sys

msg = u'Chcesz zobaczyć demo?'
title = u'Wybierz opcję'
if eg.ccbox(msg, title):
      eg.egdemo()
else:
      while 1:
            # okienko typu alert
            eg.msgbox(u'Treść w okienku', u'Tytuł w okienku')

            # okienko typu select
            msg =u'Twój ulubiony kolor?'
            title = 'Kolory'
            # niestety tu bez polskich znaków:
            choices = ['bialy', 'czarny', 'niebieski', 'inny']
            choice = eg.choicebox(msg, title, choices)
```

```
            eg.msgbox("Twój ulubiony kolor to: %s" % str(choice), u'Twój
wybór')

            # okienko typu confirm
            msg = u'Chcesz kontynuować?'
            title = u'Wybierz opcję'
            if eg.ccbox(msg, title):
                  pass
            else:
                  sys.exit(0)
```

Bardziej szczegółowe informacje:

- http://easygui.sourceforge.net/tutorial/index.html#contents_item_8.1

Wyrażenia regularne

```
import re

text = 'Mam na imie Jarek, mam 16 lat.'
# proste szukanie
regex = r'\d* lat'
result = re.search(regex, text)
if result:
      result_text = result.group()
      print result_text #zwróci '16 lat'

# szukanie z grupowaniem
regex = r'(\d*) lat'
r = re.compile(regex)
result = r.search(text)
# result = r.match(text) #sprawdzałoby, czy znajduje się na początku ciągu
print result.group() #zwróci '16 lat'
print result.group(1) #zwróci '16'
```

Bardziej szczegółowe informacje:

- http://home.agh.edu.pl/~mkuta/tk/re/re.html
- http://ftp.us4us.eu/EDU/2013/Python-dla-naukowca/Python-2013-03.pdf

Parsowanie XML

```
from BeautifulSoup import BeautifulSoup

x="""<foo>
   <bar>
      <type foobar="1"/>
      <type foobar="2"/>
   </bar>
</foo>"""
y=BeautifulSoup(x)
print y.foo.bar.type["foobar"] #wyświetli '1' - pierwszy pasujący element
print y.foo.bar.findAll("type") #wyszuka wszystkie elementy type - tu 2
print y.foo.bar.findAll("type")[0]["foobar"] #wyświetli '1'
print y.foo.bar.findAll("type")[1]["foobar"] #wyświetli '2'
```

Generowanie XML

```
from lxml import etree

root = etree.Element('root')
root.append(etree.Element('child'))
```

```
child = etree.Element('child')
child.text = 'some text'
child.set('artybut', 'wartosc')
root.append(child)

s = etree.tostring(root, pretty_print=True)
print s
```

Bardziej szczegółowe informacje:

- http://lxml.de/tutorial.html

Użycie szablonów (biblioteka Cheetah)

```
from Cheetah.Template import Template

template = """<HTML>
<HEAD><TITLE>$title</TITLE></HEAD>
<BODY>
$contents
## komentarz jednoliniowy
#*
komentarz
wieloliniowy
*#
$contents2
#if $if
      Warunek spelniony
#else
      Warunek niespelniony
#end if
#for $item in $for
      wartosc: $item
#end for
</BODY>
</HTML>"""

nameSpace = {'title': 'Tytul strony', 'contents': 'Hello World!'}
nameSpace['contents2'] = 'Tresc2!'
nameSpace['if'] = True
# nameSpace['if'] = False
nameSpace['for'] = ['one', 'two', 'three']
t = Template(template, searchList=[nameSpace])
print t
```

Rozwiązywanie problemów z polskimi znakami

Rozwiązywanie problemów na przykładzie obsługi biblioteki Cheetah:

```
#-*- coding: utf-8 -*-

from Cheetah.Template import Template

template = u"""#encoding UTF-8
<HTML>
<HEAD><TITLE>$title</TITLE></HEAD>
<BODY>
$contents
## komentarz jednoliniowy
#*
komentarz
```

```
wieloliniowy
*#
$contents2
#if $if
      Warunek spelniony
#else
      Warunek niespelniony
#end if
#for $item in $for
      wartosc: $item
#end for
</BODY>
</HTML>"""

nameSpace = {'title': u'Tytuł strony', 'contents': 'Hello World!'}
nameSpace['contents2'] = 'Tresc2!'
nameSpace['if'] = True
# nameSpace['if'] = False
nameSpace['for'] = ['one', 'two', 'three']
t = Template(template, searchList=[nameSpace])
t = str(t).decode('utf-8')
print t

plik = open('plik.html', 'w')
plik.write(t.encode('utf-8'))
plik.close()
```

Uruchamianie programów w Pythonie

Programy w Pythonie możemy uruchamiać na wiele sposobów. Poznamy tylko kilka przykładów.

Linux

Standardowe wywołanie skryptu w Pythonie w systemie Linux wygląda następująco:

```
python skrypt.py
```

Jest możliwość uruchomienia skryptu jak zwykłego programu. W pierwszej (koniecznie) linijce skryptu należy umieścić kod:

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

W tym wypadku skrypt taki możemy uruchomić poprzez wydanie polecenia:

```
./skrypt.py
```

Plik taki oczywiście musi mieć uprawnienia do wykonania (chmod u+x).

Windows

W systemie Windows w konsoli standardowo uruchamiamy skrypt analogicznie do systemu Linux:

```
python skrypt.py
```

Musimy jednak pamiętać o tym, aby ścieżka C:\python (przykładowa) znalazła się w zmiennej środowiskowej PATH.

Jeśli jednak ustawiliśmy domyślny program obsługujący na interpreter Pythona, możemy wywołać po prostu nazwę pliku:

```
skrypt.py
```

W tym wypadku możemy uruchomić skrypt także ze środowiska typowo okienkowego. Jeśli jednak skrypt nie ma zatrzymania, to prawdopodobnie uruchomi się, wykona i zniknie zanim zdążymy coś zobaczyć (w zależności od systemu).

Inny język programowania

Większość (jak nie wszystkie) języków ma jakieś narzędzie do odpalania programów zewnętrznych. Jako przykład podam kod w PHP, który uruchomi skrypt napisany w Pythonie:

```
<?php
      $zwrotka = exec('skrypt.py');
      var_dump($zwrotka);
?>
```

Wersja wykonywalna programu

Ponieważ do systemu Linux standardowo dostarczony jest Python, nie musimy martwić się, że odbiorca nie uruchomi naszego skryptu. Sytuacja jest bardziej skomplikowana w przypadku Windowsa. Tutaj są 2 możliwości:

- dostarczenie interpretera Pythona wraz z instalacja naszego programu (robi tak np. OpenOffice)
- stworzenie wersji exe

My zajmiemy się tą drugą opcją. Do tego celu będzie nam potrzebny pakiet py2exe oraz skrypt:

```
#-*- coding: utf-8 -*-
from distutils.core import setup
import py2exe
setup(
    version = "1.0",
    console = ["testy.py"],
    )
```

Zamiast testy.py wpisujemy nazwę naszego pliku, a następnie zapisujemy (np. pod nazwą compile.py). Następnie z konsoli wykonujemy polecenie:

```
python compile.py py2exe
```

W efekcie w katalogu ze skryptem zostaną utworzone 2 nowe katalogi. W katalogu dist znajduje się plik exe, oraz biblioteki potrzebne do jego wykonania. Tą wersję możemy już bez przeszkód rozpowszechniać.