KORISNIČKA SUČELJA

Aleksandar Maksimović
IRB

Python

- Python interpreter

Python je interpreterski, interaktivni, objektno orjentirani programski jezik, kojeg je 1990. godine razvio Guido van Rossum. Već do konca 1998., Python je imao bazu od 300.000 korisnika, a od 2000. već su ga prihvatile ustanove kao MIT, NASA, IBM, Google, Yahoo i druge.
Python je besplatan (za akademske ustanove i neprofitnu upotrebu), open-source software, s izuzetno dobrom potporom, literaturom i dokumentacijom. http://www.python.org

Python

Interpretacija međukôda

Python kôd živi u tekst datotekama koje završavaju na *.py*. Program kompilira kôd u niz *bytecode-ova* koji se spremaju u *.pyc* datoteke koje su prenosive na bilo koje platforme gdje se mogu izvoditi interpretacijom tog međukôda. Slično se izvršava Java kôd. Brzina izvođenja Python kôda istog je reda veličine kao u Javi ili Perlu. Python je napisan u ANSI C i raspoloživ za cijeli niz strojeva i operacijskih sustava uključujući Windows, Unix/Linux i Macintosh.

Jezik visoke razine

Osim standardnih tipova podataka (brojevi, nizovi znakova i sl.) Python ima ugrađene tipove podataka visoke razine kao što su liste, n-terci i rječnici.

Interaktivnost

Python se može izvoditi u različitim okruženjima. Za razvitak programa najbolji je interaktivni način rada u kojem se programski kôd piše naredbu za naredbom. Ne postoji razlika u razvojnom i izvedbenom (engl. runtime) okolišu.

Python

Čista sintaksa

Sintaksa jezika je jednostavna i očevidna. Uvlake zamjenjuju posebne znakove za definiranje blokova kôda, pa je napisani program vrlo pregledan i jednostavan za čitanje.

Napredne značajke jezika

Python nudi sve značajke očekivane u modernom programskom jeziku: objektu orijentirano programiranje s višestrukim nasljeđivanjem, dohvaćanje izuzetaka, redefiniranje standardnih operatora, pretpostavljene argumente, prostore imena i pakete.

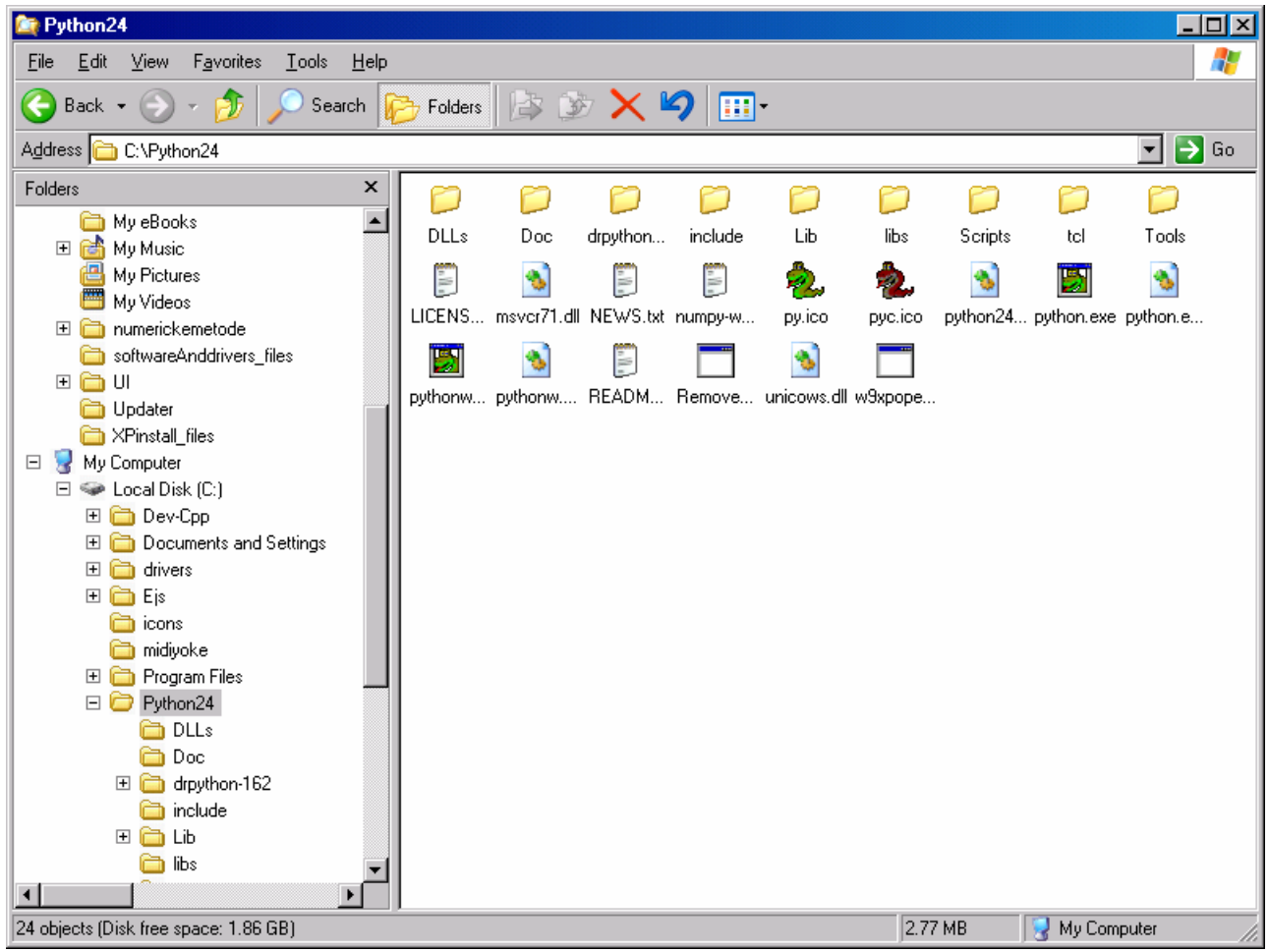
Proširivost

Python je pisan u modularnoj C arhitekturi. Zato se može lako proširivati novi značajkama ili API-ima. (eng. application programming interface).

python

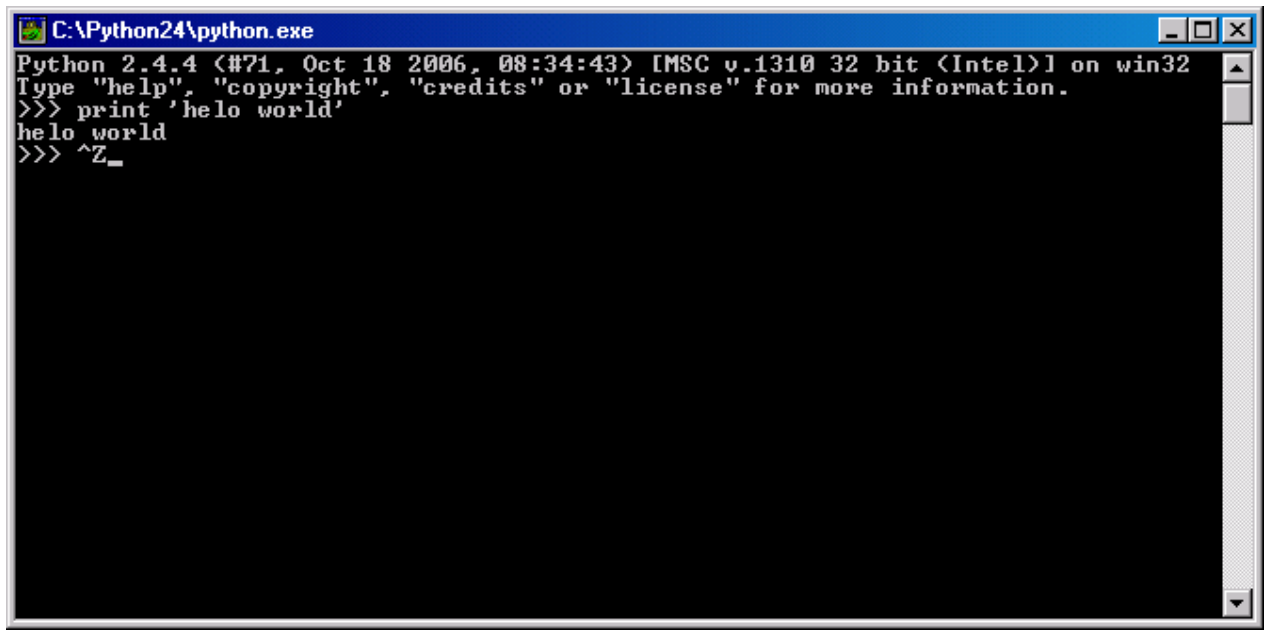
http://www.python.org

http://www.activestate.com



python interpreter

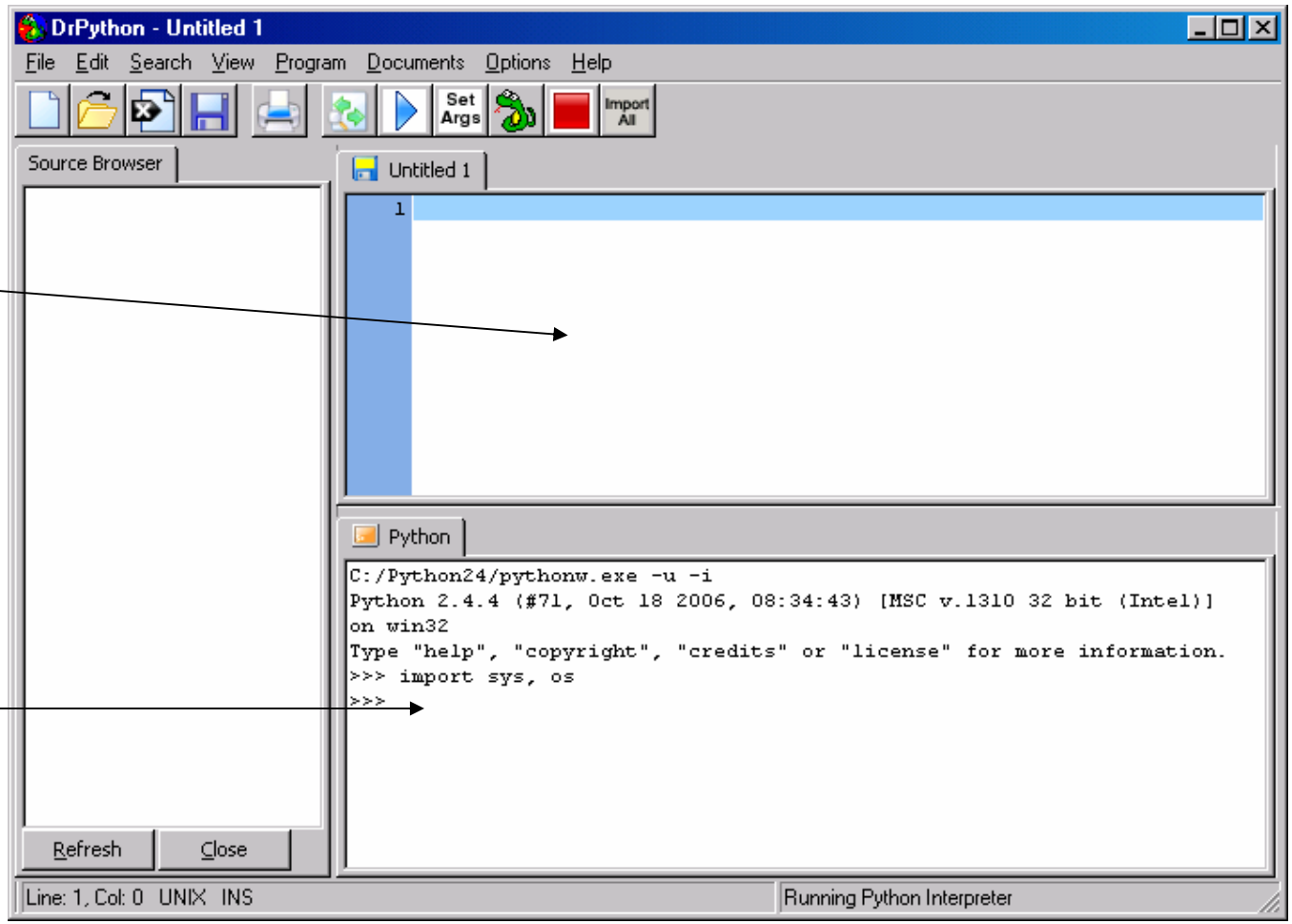
python.exe



drPython

editor

interpreter



Aleksandar Maksimović,
Institut Ruđer Bošković, Bijenička c. 54

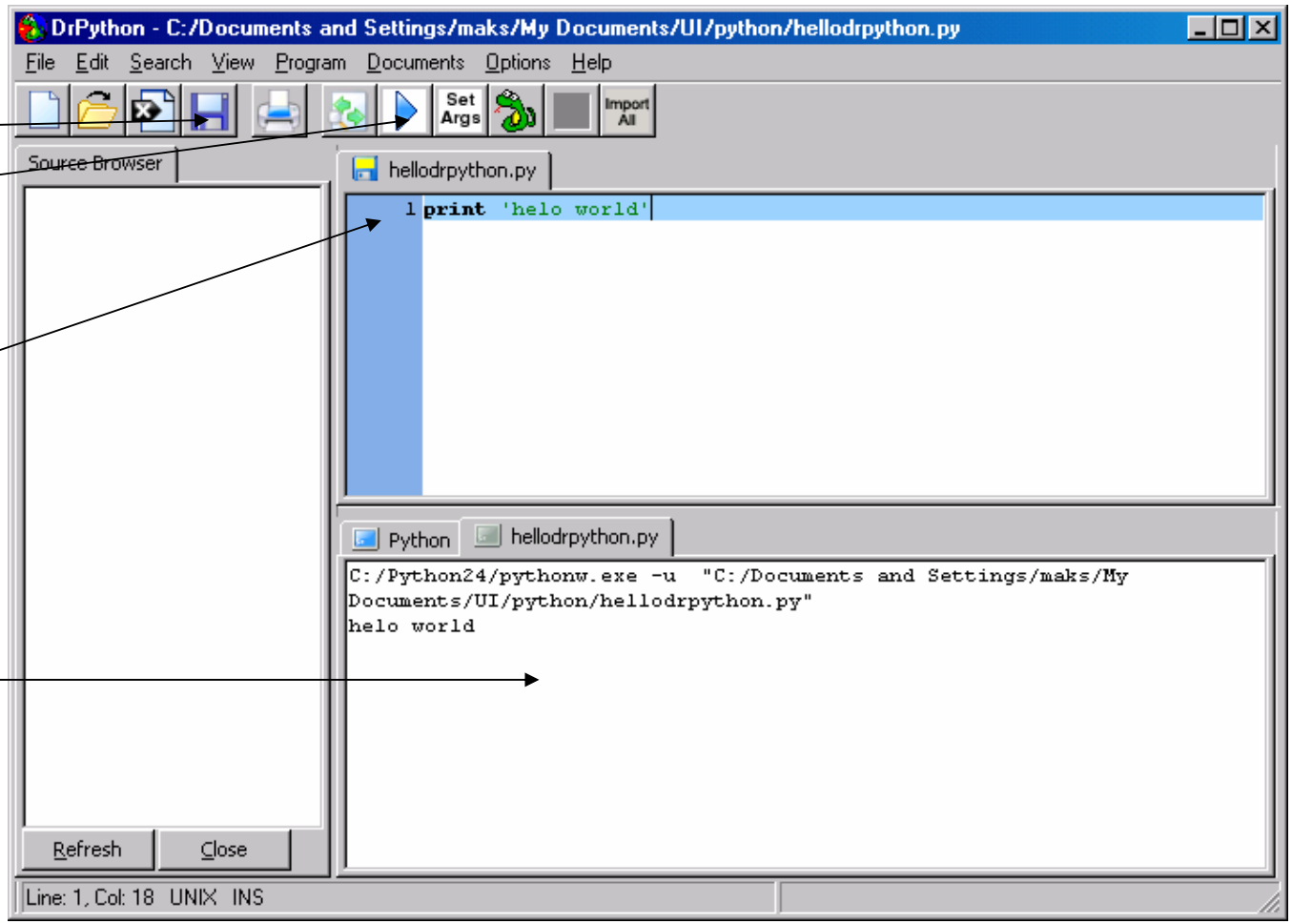
drPython

save

run

skripta

izvršena skripta



programiranje

- Konstante
  - 1, 1.23, 5.2e-3
  - 'string', "ovo je' string"
- Brojevi
  - 1,2,3 - cijeli brojevi
  - 1.2, 5.3e-3 - realni brojevi
  - (1+2j) - kompleksni brojevi

- String
  - ' - jednostruki navodnici
  - " - dvostruki
  - ''' - trostruki

'''This is a multi-line string. This is the first line.

This is the second line.

"What's your name?," I asked.

He said "Bond, James Bond."

- **Escape Sequences**
  - \' - prikazuje ', npr. 'What\'s your name?'
    - 'What's your name?' - ne bi znao gdje je početak i kraj
  - \n - nova linija
    - This is the first line\nThis is the second line
  - \t - tab
  - \ - na kraju linije nastavlja string na novu liniju

    "This is the first sentence.\

    This is the second sentence."

python

- Varijable
  - Imena
    - Prvi znak mora biti slovo ili _ (underscore)
    - Ostali znakovi u imenu varijable - slova, brojevi ili _
    - Velika i mala slova, varijable Var i var nisu iste
  - Dobra imena
    - i, __my_name, name_23
  - Loša imena
    - 2things, this is spaced out, moje-ime

primjer

```
# Filename : var.py
i = 5
print i
i = i + 1
print i
s = '''This is a multi-line string.
This is the second line.'''
print s
```

```
i = 5;
print i;
```

```
i = 5; print i;
```

```
i = 5; print i
```

```
s = 'This is a string. \
This continues the string.'
print s
```

tipovi podataka

- varijable sadrže različite vrijednosti - tipove podataka
  - Provjera tipa nekog objekta ostvaruje se pozivom funkcije type():

```
>>> k=2.178   # varijabla k
>>> type(k)
<type 'float'>
>>> type ({ 'a':72, 'b':1.4 })
<type 'dict'>
>>> z=2+3j    # varijabla z
>>> type(z)
<type 'complex'>
```

tipovi podataka

- N-terac (eng. tuples)
  - N-terac je niz objekata, istih ili različitih tipova. N-terac se definira nabrajanjem objekata odvojenih zarezima (,). Zadnjem članu u nizu takodjer se može dodati zarez. N-terac sa samo jednim članom mora imati zarez na kraju, jer inače gubi tip n-terca.

```
(100, 200, 300)        # N-terac s tri člana
(3.14,)                # N-terac sa samo jednim članom
(    )                 # Prazan n-terac
```

```
>>> x='abrakadabra'
>>> tuple(x)
('a', 'b', 'r', 'a', 'k', 'a', 'd', 'a', 'b', 'r', 'a')
```

tuples

```
>>> rec = ('Smith','John',(6,23,68))    # This is a tuple
>>> lastName,firstName,birthdate = rec  # Unpacking the tuple
>>> print firstName
John
>>> birthYear = birthdate[2]
>>> print birthYear
68
>>> name = rec[1] + ' ' + rec[0]
>>> print name
John Smith
>>> print rec[0:2]
('Smith', 'John')
```

Liste

Članovi u listi su bilo kakvi objekti različitih vrsta. Lista se definira nabrajanjem članova, odijeljenih zarezima (,) i smještenih unutar uglatih zagrada ([ ]). Dopušteno je iza zadnjeg člana liste, ostaviti još ejdan zarez. Prazna lista se označava praznim parom uglatih zagrada.

```
>>> a = [1.0, 2.0, 3.0]      # Create a list
>>> a.append(4.0)            # Append 4.0 to list
>>> print a
[1.0, 2.0, 3.0, 4.0]
```

```
>>> a = [1.0, 2.0, 3.0]
>>> b = a                    # 'b' is an alias of 'a'
>>> b[0] = 5.0               # Change 'b'
>>> print a
[5.0, 2.0, 3.0]              # The change is reflected in 'a'
>>> c = a[:]                 # 'c' is an independent copy of 'a'
>>> c[0] = 1.0               # Change 'c'
>>> print a
[5.0, 2.0, 3.0]              # 'a' is not affected by the change
```

pridružuje se "by reference" svaka promjena u novoj varijabli, mijenja početnu listu

"by value"

Liste

- Matrice možemo prikazati pomoću lista

```
>>> a = [[1, 2, 3], \
         [4, 5, 6], \
         [7, 8, 9]]
>>> print a[1]          # Print second row (element 1)
[4, 5, 6]
>>> print a[1][2]       # Print third element of second row
6
```

Operacije na nizovima

Dohvaćanje elementa bilo kojeg niza (stringa, n-terca, liste) postiže se indeksiranjem. Dio niza, odlomak ili kriška (engl. slice) dobiva se sintaksom 'i:j' gdje je 'i' početni indeks, a 'j' završni indeks kriške (tablica 2.4.). Dužina niza dobiva se pozivom funkcije len(), a maksimalni i minimalni član niza s funkcijama max(), odnosno min().

Član	Opis
`s[i]`	Vraća element *i* u nizu *s*
`s[i:j]`	Vraća kršku – niz elemenata od i-tog do j-tog indeksa
`len(s)`	Vraća broj elemenata u *s*
`min(s)`	Vraća minimalni elemenat iz *s*
`max(s)`	Vraća maksimalni elemenat iz *s*

Operacije na nizovima

Član	Opis
`s [i ] = v`	Pridružba članu na i-tom mjestu
`s [i :j ] = t`	Pridružba skupini članova
`del s [i ]`	Brisanje člana
`del s [i :j ]`	Brisanje skupine članova

```
>>> a=(1,3,5,7,9)
>>> print a[0], a[3]
1 7
>>> b='ovo je string'
>>> print b[9],b[0],b[-1]
r o g
```

vrijednosti niza, stringa

promjena vrijednosti u listi

```
>>> lista=['tko zna','bilje','siroko mu','polje']
>>> lista[1]='bolje'
>>> lista
['tko zna', 'bolje', 'siroko mu', 'polje']
>>> lista[1:3]=[',zna!']
```

Rječnik

Preslikavanje (engl. mapping) je skup objekata indeksiranih s pomoću gotovo slobodnih vrijednosti koje se zovu ključevi (engl. keys). Tako nastali objekti su promjenljivi, a za razliku od nizova, nisu poredani. Eksplicitno stvaranje rječnika provodi se nizom parova ključ:vrijednost odvojenih zarezima, koji se smještaju unutar vitičastih zagrada.

```
{'x':42, 'y':3.14, 'z':7}  # Rječnik s tri člana i string ključevima
{1: 2, 3:4}                # Rječnik s dva člana i cjelobrojnim ključevima
{ }                        # Prazan rječnik
```

```
>>> dict([['a',12],['b',54]])
{'a': 12, 'b': 54}
>>> dict(a='zagreb', d='ogulin', e='Osijek')
{'a': 'zagreb', 'e': 'Osijek', 'd': 'ogulin'}
```

metode - string

Metoda	Opis
`s.capitalize()`	Pretvara po slovo od *s* u veliko slovo.
`s.center(width)`	Centrira string upolju duljine *width*.
`s.count(sub[,start[,end]])`	Broji pojavljivanja podstringa *sub* u stringu *s*.
`s.encode([encoding[,errors]])`	Vraća kodiranu inačicu stringa
`s.endswith(suffix[,start[,end]])`	Provjerava kraj stringa za suffix.
`s.expandtabs([tabsize])`	Proširuje tabulatore praznim mjestima
`s.find(sub[,start[,end]])`	Pronalazi prvo pojavljivanje zadanog podstringa *sub*.
`s.index(sub[,start[,end]])`	Pronalazi prvo pojavljivanje zadanog podstringa *sub* uz podizanje izuzetka, ako ga nema
`s.isalnum()`	Provjerava jesu li svi znakovi alfanumerički
`s.isalpha()`	Provjerava jesu li svi znakovi alfabetski.
`s.isdigit()`	Provjerava jesu li svi znakovi znamenke.
`s.islower()`	Provjerava jesu li svi znakovi pisani malim slovima.
`s.isspace()`	Provjerava jesu li svi znakovi praznine.

metode - string

`s.isupper()`	Provjerava jesu li svi znakovi pisani velikim slovima.
`s.join(t)`	Povezuje stringove u listi `t` koristeći `s` kao međaš.
`s.ljust(width)`	Lijevo poravnanje `s` u stringu duljine `width`.
`s.lower()`	Vraća `s` pretvoren u string s malim slovima.
`s.lstrip()`	Odstranjuje prazna mjesta ispred stringa.
`s.replace(old, new[,maxreplace])`	Zamjenjuje podstring `old` sa `new`.
`s.rfind(sub[,start[,end]])`	Nalazi zadnji pojavak podstringa `sub`
`s.rindex(sub[,start[,end]])`	Nalazi zadnji pojavak podstringa `sub` ili javlja izuzetak
`s.rjust(width)`	Desno poravnanje `s` u stringu duljine `width`.
`s.rstrip()`	Odstranjuje prazna mjesta iza stringa.
`s.split([sep[,maxsplit]])`	Dijeli string koristeći `sep` kao međaš. `maxsplit` je naveći broj dijeljenja koji će se izvršiti.
`s.splitlines([keepends])`	Dijeli string u listu linija. Ako je `keepends` jednak `1`, čuvaju se kontrolni znakovi novih redaka.
`s.startswith(prefix[,start[,end]])`	Provjerava da li string započinje s `prefix`-om .
`s.strip()`	Odstranjuje prazna mjesta i ispred i iza stringa.
`s.swapcase()`	Vraća velika slova za string malih sloa i obratno.

metode - liste

Metoda	Opis
`li.append(x)`	Dodaje novi element x na kraj liste `li`.
`li.extend(t)`	Dodaje novu listu `t` na kraj liste `li`.
`li.count(x)`	Broji pojavke od `x` u listi `li`.
`li.index(x)`	Vraća najmanji `i` za koji je `s[i] == x`.
`li.insert(i,x)`	Umeće `x` na indeksu `i`.
`li.pop([i])`	Vraća element `i` i briše ga iz liste. Ako se `i` izostavi, onda se vraća zadnji element.
`li.remove(x)`	Traži `x` i briše ga iz liste `li`.
`li.reverse()`	Reverzira (obrće) članove liste `li` na mjestu.
`li.sort([cmpfunc])`	Sortira (slaže) članove liste `li` na mjestu. `cmpfunc` je funkcija za usporedbu

metode - rječnik

Član	Opis
`di [k ]`	Vraća član od `di` s ključem `k` .
`di [k ] = x`	Postavlja `di [k ]` na `x` .
`del di [k ]`	Briše `di [k ]` iz `di` .
`di .clear ()`	Briše sve članove iz `di` .
`di .copy ()`	Vraća kopiju od `di` .
`di .has_key (k )`	Vraća `1` ako `di` ima ključ `k` , a `0` inače.
`di .items ()`	Vraća listu od `(key , value )` parova.
`di .keys ()`	Vraća listu od vrijednosti ključeva.
`di .update (b )`	Dodaje sve objekte iz rječnika `b` u `di` .
`di .values ()`	Vraća listu svih vrijednosti spremljenih u `di` .
`di .get (k [, v ])`	Vraća `di [k ]` ako nađe; inače vraća `v` .
`di .setdefault (k [, v ])`	Vraća `di [k ]` ako nađe; vraća `v` ipostavlja `di [k ]=v` .
`di .popitem ()`	Vraća slučajne `(key , value)` parove kao e-torke iz `di`.

`di.has_key(k)`	Vraća `1` ako `di` ima ključ `k`, a `0` inače.
`di.items()`	Vraća listu od `(key,value)` parova.
`di.keys()`	Vraća listu od vrijednosti ključeva.
`di.update(b)`	Dodaje sve objekte iz rječnika `b` u `di`.
`di.values()`	Vraća listu svih vrijednosti spremljenih u `di`.
`di.get(k[,v])`	Vraća `di[k]` ako nađe; inače vraća `v`.
`di.setdefault(k[, v])`	Vraća `di[k]` ako nađe; vraća `v` ipostavlja `di[k]=v`.
`di.popitem()`	Vraća slučajne `(key,value)` parove kao e-torke iz `di`.

import

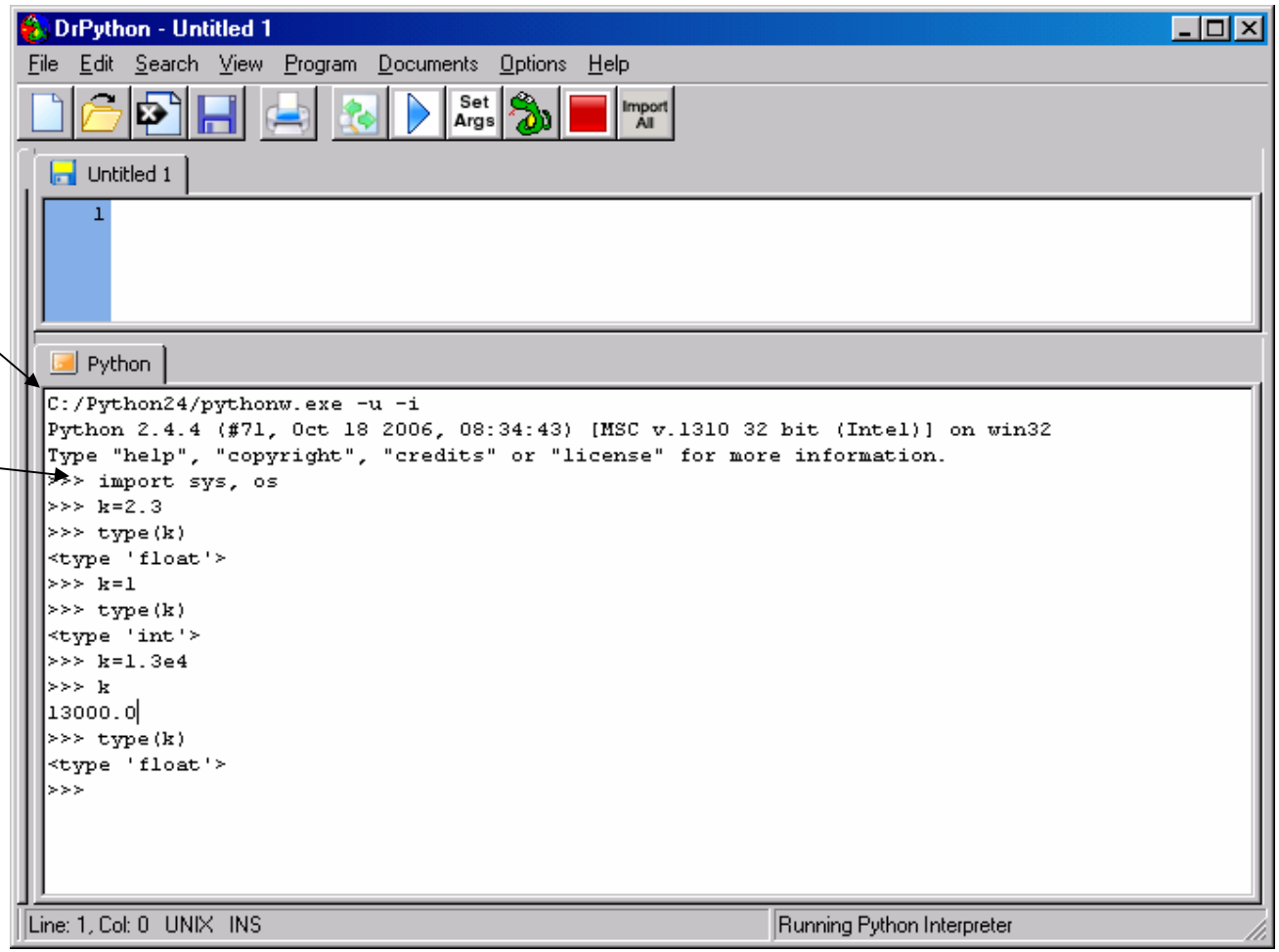
interpreter komanda

import sys -učitava se modul sistemskih funkcija

Pojedinačni podatak ili funkcija iz modula dohvaća se naredbom 'from ... import ... ' :

from math import sin, cos

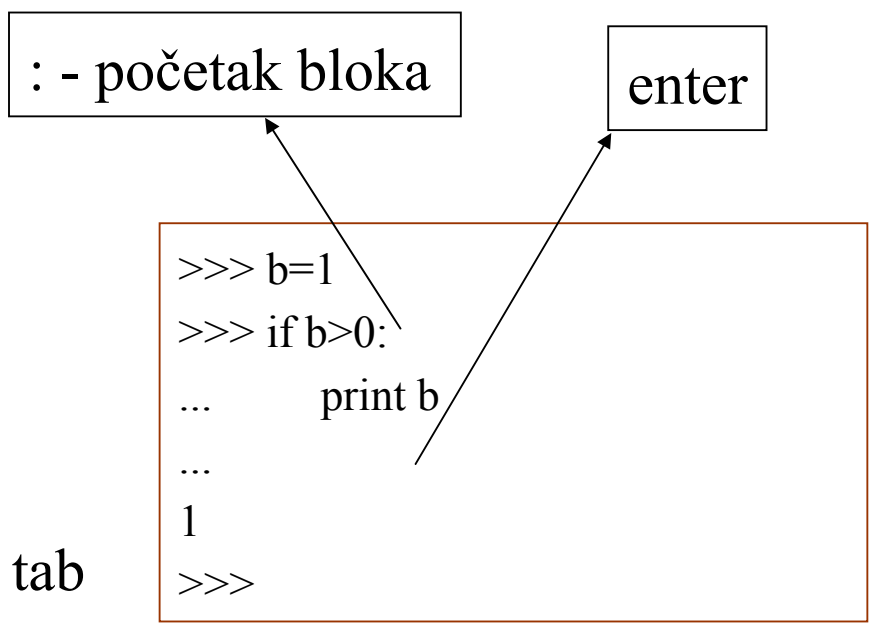
funkcije sin i cos iz math modula



programiranje

- indent - uvlačenje retka koristimo kod naredbi toka
  - prazni (bijeli) prostor (whitespace= tab ili space) na početku linije označava kod u istom bloku, npr.

```
>>> print "bok"
bok
>>>  print "bok"
  File "<stdin>", line 1
    print "bok"
    ^
SyntaxError: invalid syntax
```



- Kod If komande koristimo whitespace - tab

Operatori

- Kalkulator - možemo interaktivno zbrajati, oduzimati, itd..

Operator	Name	Explanation	Examples
+	Plus	Adds the two objects	`3 + 5` gives `8`. `'a' + 'b'` gives `'ab'`.
-	Minus	Either gives a negative number or gives the subtraction of one number from the other	`-5.2` gives a negative number. `50 - 24` gives `26`.
*	Multiply	Gives the multiplication of the two numbers or returns the string repeated that many times.	`2 * 3` gives `6`. `'la' * 3` gives `'lalala'`.
**	Power	Returns x to the power of y	`3 ** 4` gives `81` (i.e. `3 * 3 * 3 * 3`)
/	Divide	Divide x by y	`4/3` gives `1` (division of integers gives an integer). `4.0/3` or `4/3.0` gives `1.3333333333333333`

Operatori

```
>>> s = 'Hello '
>>> t = 'to you'
>>> a = [1, 2, 3]
>>> print 3*s                  # Repetition
Hello Hello Hello
>>> print 3*a                  # Repetition
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> print a + [4, 5]           # Append elements
[1, 2, 3, 4, 5]
```

Neki od operatora vrijede za stringove

definirani su skraćeni operatori kao u C jeziku

a += b	a = a + b
a -= b	a = a - b
a *= b	a = a*b
a /= b	a = a/b
a **= b	a = a**b
a %= b	a = a%b

Operatori

<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to

1 - istinita tvrdnja

0 - neistinita tvrdnja

Uspoređivanje vrijednosti i stringova

```
>>> a = 2              # Integer
>>> b = 1.99           # Floating point
>>> c = '2'            # String
>>> print a > b
1
>>> print a == c
0
>>> print (a > b) and (a != c)
1
>>> print (a > b) or (a == b)
1
```

Expressions

- Izrazi - linije koda u kojima pridružujemo vrijednosti i/ili koristimo operatore kako bi dobili novu vrijednost

```
# Filename: expression.py

length = 5
breadth = 2

area = length * breadth
print 'Area is', area
print 'Perimeter is', 2 * (length + breadth)
```

Kontrola toka

if - komanda

```
if condition:
    block
```

možemo je koristiti zajedno s jednom ili više dodatnih uvjeta

```
elif condition:
    block
```

ako nije nijedan uvjet ispunjen koristimo *else* komandu

```
else:
    block
```

blokovi *elif* i *else* nisu obavezni

kontrola toka

```
#!/usr/bin/python
# Filename: if.py

number = 23
guess = int(raw_input('Enter an integer : '))

if guess == number:
        print 'Congratulations, you guessed it.' # New block starts here
        print "(but you do not win any prizes!)" # New block ends here
elif guess < number:
        print 'No, it is a little higher than that' # Another block
        # You can do whatever you want in a block ...
else:
        print 'No, it is a little lower than that'
        # you must have guess > number to reach here

print 'Done'
# This last statement is always executed, after the if statement is executed
```

Petlje (Loops)

```
while condition:
    block
```

izvršava blok sve dok je zadovoljen uvjet *condition*

kao i kod *if* komande možemo koristiti *else* komandu ako uvjet nije istinit

```
else:
    block
```

Petlje

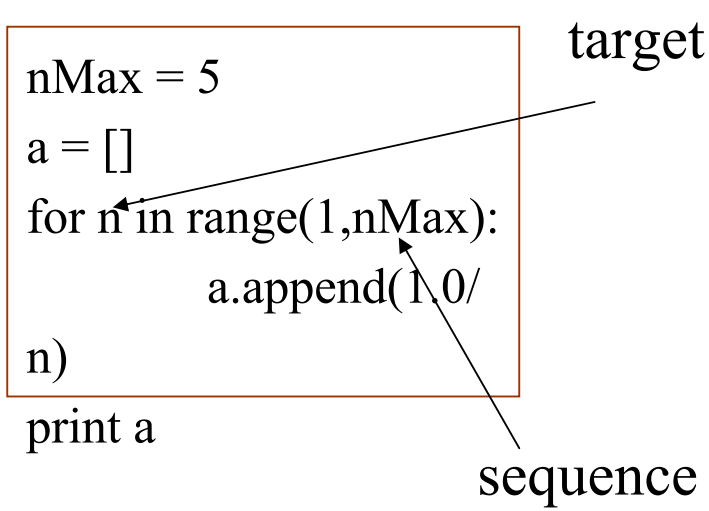
```
nMax = 5
n = 1
a = []                  # Create empty list
while n < nMax:
    a.append(1.0/n)     # Append element to list
    n = n + 1
print a
```

Rezultat je: `[1.0, 0.5, 0.3333333333333331, 0.25]`

Petlje

Izvršava se blok naredbi dok se vrijednost *target* nalazi u nizu *sequence*

```
for target in sequence:
    block
```




```
list = ['Jack', 'Jill', 'Tim', 'Dave']
name = eval(raw_input('Type a name: ')) # input
for i in range(len(list)):
        if list[i] == name:
                print name,'is number',i + 1,'on the
list'
                break
        else:
                print name,'is not on the list'
```

Kontrola pogreške

Pogreške prilikom izvođenja programa mogu se uhvatiti naredbama *try* i *except*

```
try:
    do something
except error:
    do something else
```

```
>>> c = 12.0/0.0
Traceback (most recent call last):
  File ''<pyshell#0>'', line 1, in ?
    c = 12.0/0.0
ZeroDivisionError: float division
```

This error can be caught by

```
try:
    c = 12.0/0.0
except ZeroDivisionError:
    print 'Division by zero'
```

break & continue

break i *continue* modificiraju izvršavanje petlje. *break* omogućava prekidanje petlje

```
f = open(filename, 'r')
while 1:
    line = f.readline()
    if line == '':        # empty string means end of file
        break             # jump out of while loop
    # process line
    ...
```

continue omogućava da skočimo na slijedeću iteraciju

```
files = os.listdir(os.curdir)  # all files/dirs in current dir.
for file in files:
    if not os.path.isfile(file):
        continue   # not a regular file, continue with next
    <process file>
```

Funkcije

- Fukncije definiramo pomoću komande *def*

```
def func_name(param1, param2,...):
    statements
    return return_values
```

primjer: 1 i 2-ga derivacija

```
from math import arctan
def  finite_diff(f,x,h=0.0001):    # h has a default value
    df =(f(x+h) - f(x-h))/(2.0*h)
    ddf =(f(x+h) - 2.0*f(x) + f(x-h))/h**2
    return df,ddf
x = 0.5
df,ddf = finite_diff(arctan,x)    # Uses default value of h
print 'First derivative  =',df
print 'Second derivative =',ddf
```

def, module

Ako je argument lista, onda se vrijednost liste mijenja i u glavnom programu

```
def squares(a):
    for i in range(len(a)):
        a[i] = a[i]**2

a = [1, 2, 3, 4]
squares(a)
print a
```

The output is

```
[1, 4, 9, 16]
```

```
from module_name import *
```

module math

```
from math import *
```

samo odabrane funkcije

```
from math import func1, func2,...
```

module, doc string

```
>>> import math
>>> dir(math)
['__doc__', '__name__', 'acos', 'asin', 'atan',
 'atan2', 'ceil', 'cos', 'cosh', 'e', 'exp', 'fabs',
 'floor', 'fmod', 'frexp', 'hypot', 'ldexp', 'log',
 'log10', 'modf', 'pi', 'pow', 'sin', 'sinh', 'sqrt',
 'tan', 'tanh']
```

doc string - nakon definicije funkcije string s trostrukim navodnicima

```
def mkdir(dir, mode=0777, remove=True, chdir=True):
    """
    Create a directory dir (os.mkdir(dir,mode)).
    If dir exists, it is removed by shutil.rmtree if
    remove is true. If chdir is true, the current working
    directory is set to dir (os.chdir(dir)).
    """
    ...
```

Funkcije

promjenljivi broj argumenata specificira se *

```
def somefunc(a, b, *args):
    # args is a tuple of all supplied positional arguments
    ...
    for arg in args:
        <work with arg>
```

** kada je argument dictionary promjenljive duljine

```
def somefunc(a, b, *args, **kwargs):
    # args is a tuple of all supplied positional arguments
    # kwargs is a dictionary of all supplied keyword arguments
    ...
    for arg in args:
        <work with arg>
    for key in kwargs:
        <work with argument key and its value kwargs[key]>
```

Funkcije

* - promjenjljivi broj argumenata, ne znači da argument ne može biti lista kojoj ne znamo duljinu unaprijed

```
>>> aa=[1,2,3]
>>> def myf(bb):
...         print bb[1]
...         bb[1]=0
...
>>> myf(aa)
2
```

1 argument

```
>>> def mz(*bl):
...         print bl
...
>>> mz(aa)
([1, 0, 3],)
```

3 argumenta

```
>>> mz(1,(2,3),4)
1
(2, 3)
4
>>>
```

Funkcije

Brojevi, N-terci (tuples) ne možemo mjenjati vrijednost u funkciji

```
>>> bb=(1,2,3)
>>> myf(bb)
2
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "<stdin>", line 3, in myf
TypeError: object does not support item assignment
```

```
>>> def swap(a, b):
        tmp = b; b = a; a = tmp;

>>> a = 1.2; b = 1.3;
>>> swap(a, b)
>>> a, b    # has a and b been swapped?
(1.2, 1.3)  # no...
```

kao u C jeziku (by value):

```
>>> def swap(a, b):
       return b, a   # return tuple (b, a)

>>> a = 1.2; b = 1.3;
>>> a, b = swap(a, b)
>>> a, b    # has a and b been swapped?
(1.3, 1.2)  # yes!
```

lokalne varijable

```
#!/usr/bin/python
# Filename: func_local.py

def func(x):
        print 'x is', x
        x = 2
        print 'Changed local x to', x

x = 50
func(x)
print 'x is still', x
```

output:

```
x is 50
Changed local x to 2
x is still 50
```

global

global statement

```
# Filename: func_global.py

def func():
        global x

        print 'x is', x
        x = 2
        print 'Changed global x to', x

x = 50
func()
print 'Value of x is', x
```

output:

```
x is 50
Changed global x to 2
Value of x is 2
```

Funkcije

Default argumenti

```
#!/usr/bin/python
# Filename: func_default.py

def say(message, times = 1):
        print message * times

say('Hello')
say('World', 5)
```

Output

```
Hello
WorldWorldWorldWorldWorld
```

Funkcije

Keyword argumenti

```
# Filename: func_key.py

def func(a, b=5, c=10):
        print 'a is', a, 'and b is', b, 'and c is', c

func(3, 7)
func(25, c=24)
func(c=50, a=100)
```

Output

```
a is 3 and b is 7 and c is 10
a is 25 and b is 5 and c is 24
a is 100 and b is 5 and c is 50
```

Funkcije

return - vrati željenu vrijednost i prekine izvršavanje funkcije

```
# Filename: func_return.py

def maximum(x, y):
        if x > y:
                return x
        else:
                return y

print maximum(2, 3)
```

Output

```
$ python func_return.py
3
```

main ili module

Funkcija kao glavni program ili module dinamički određeno

```
# Filename: using_name.py

if __name__ == '__main__':
        print 'This program is being run by itself'
else:
        print 'I am being imported from another module'
```

```
$ python using_name.py
This program is being run by itself

$ python
>>> import using_name
I am being imported from another module
>>>
```

Funkcije/modul

```
#!/usr/bin/python
# Filename: mymodule.py

def sayhi():
        print 'Hi, this is mymodule speaking.'

version = '0.1'

# End of mymodule.py
```

```
#!/usr/bin/python
# Filename: mymodule_demo.py

import mymodule

mymodule.sayhi()
print 'Version', mymodule.version
```

```
$ python mymodule_demo.py
Hi, this is mymodule speaking.
Version 0.1
```

Funkcije/globalne varijable

```
>>> a = 5 # create a new variable 'a'
>>> dir()
['__builtins__', '__doc__', '__name__', 'a', 'sys']
>>>
>>> del a # delete/remove a name
>>>
>>> dir()
['__builtins__', '__doc__', '__name__', 'sys']
>>>
```

Files

Otvaranje, pisanje i zatvaranje datoteka (files). Pretpostavimo da imamo podatke

```
some comment line
1.5
  measurements  model1 model2
     0.0          0.1    1.0
     0.1          0.1    0.188
     0.2          0.2    0.25
```

Program treba napraviti 3 file-a, measurment.dat, model1.dat i model2.dat. Prvi stupac se generira s korakom 1.5, a drugi stupac je zadan. Npr. model1.dat

```
0    0.1
1.5  0.1
3    0.2
```

Files

Tok programa:

1. Otvori file, ime je prvi argument prilikom pozivanja programa. Ako fali argument obavijesti korisnika.
2. Pročitaj i preskoči prvu liniju, komentar.
3. Pročitaj vremenski korak iz druge linije
4. Pročitaj imena fileova pomoću trećeg reda. Napravi listu file objekata za različite fileove.
5. Pročitaj ostatak filea, liniju po liniju, razdvoji liniju u y vrijednosti i zapiši vrijednost u odgovarajući file zajedno s generiranim vremenom.

skripta

```
#!/usr/bin/env python
import sys, math, string
usage = 'Usage: %s infile' % sys.argv[0]
try:
        infilename = sys.argv[1]
except:
        print usage; sys.exit(1)
ifile = open(infilename, 'r') # open file for reading
# read first comment line (no further use of it here):
line = ifile.readline()
# next line contains the increment in t values:
dt = float(ifile.readline())
# next line contains the name of the curves:
ynames = ifile.readline().split()
# list of output files:
outfiles = []
for name in ynames:
        outfiles.append(open(name + '.dat', 'w'))
t = 0.0 # t value
# read the rest of the file line by line:
for line in ifile:
        yvalues = line.split()
        if len(yvalues) == 0: continue # skip blank
        for i in range(len(outfiles)):
                outfiles[i].write('%12g %12.5e\n'%\
                        (t, float(yvalues[i])))
        t += dt
for file in outfiles: file.close()
```

skripta

Modifikacije na prvu verziju skripte.

Možemo cijeli file učitati u listu.

```
f = open(infilename, 'r'); lines = f.readlines(); f.close()
```

dt - vremenski korak dobivamo iz lines[1].

imena

petlja, brojevi

dictionary

```
# the third line contains the name of the time series:
ynames = lines[2].split()

# store y data in a dictionary of lists of floats:
y = {}            # declare empty dictionary
for name in ynames:
    y[name] = [] # empty list (of y values of a time series)

# load data from the rest of the lines:
for line in lines[3:]:
    yvalues = [float(x) for x in line.split()]
    if len(yvalues) == 0: continue  # skip blank lines
    i = 0  # counter for yvalues
    for name in ynames:
        y[name].append(yvalues[i]); i += 1
```

skripta

```
yvalues = [float(x) for x in line.split()]
```

Linija programa razdvaja liniju u listu stringova koje pretvara u realne brojeve primjenom funkcije float.
Zadnja petlja treba varijablu *i* koja broji o kojoj se vrijednosti radi (indeks) u listi *yyvalue*. Lijepša sintaksa je

```
for name, yvalue in zip(ynames, yvalues):
    y[name].append(yvalue)
```

gdje zip omogućuje iteriranje preko više varijabli.

zip

loop over a common index,

```
for i in range(len(xlist)):
    x = xlist[1];  y = ylist[i];  z = zlist[i]
    # or more compactly:  x, y, z = xlist[i], ylist[i], zlist[i]
    # work with x, y, and z
```

A shorter and more Pythonic alternative is to apply the `zip` function:

```
for x, y, z in zip(xlist, ylist, zlist):
    # work with x, y, and z
```

skripta

Na kraju skripte pišemo vrijednosti t i y u file

```
for name in y.keys():
    ofile = open(name+'.dat', 'w')
    for k in range(len(y[name])):
        ofile.write('%12g %12.5e\n' % (k*dt, y[name][k]))
    ofile.close()
```

convert1.py - skripta za prvu verziju

convert2.py - skripta nakon promjena

direktorij vjezbe2

Files

Otvaranje, pisanje i zatvaranje datoteka (files). Pretpostavimo da imamo podatke

```
some comment line
1.5
  measurements  model1 model2
     0.0          0.1    1.0
     0.1          0.1    0.188
     0.2          0.2    0.25
```

Program treba napraviti 3 file-a, measurment.dat, model1.dat i model2.dat. Prvi stupac se generira s korakom 1.5, a drugi stupac je zadan. Npr. model1.dat

```
0    0.1
1.5  0.1
3    0.2
```

Files

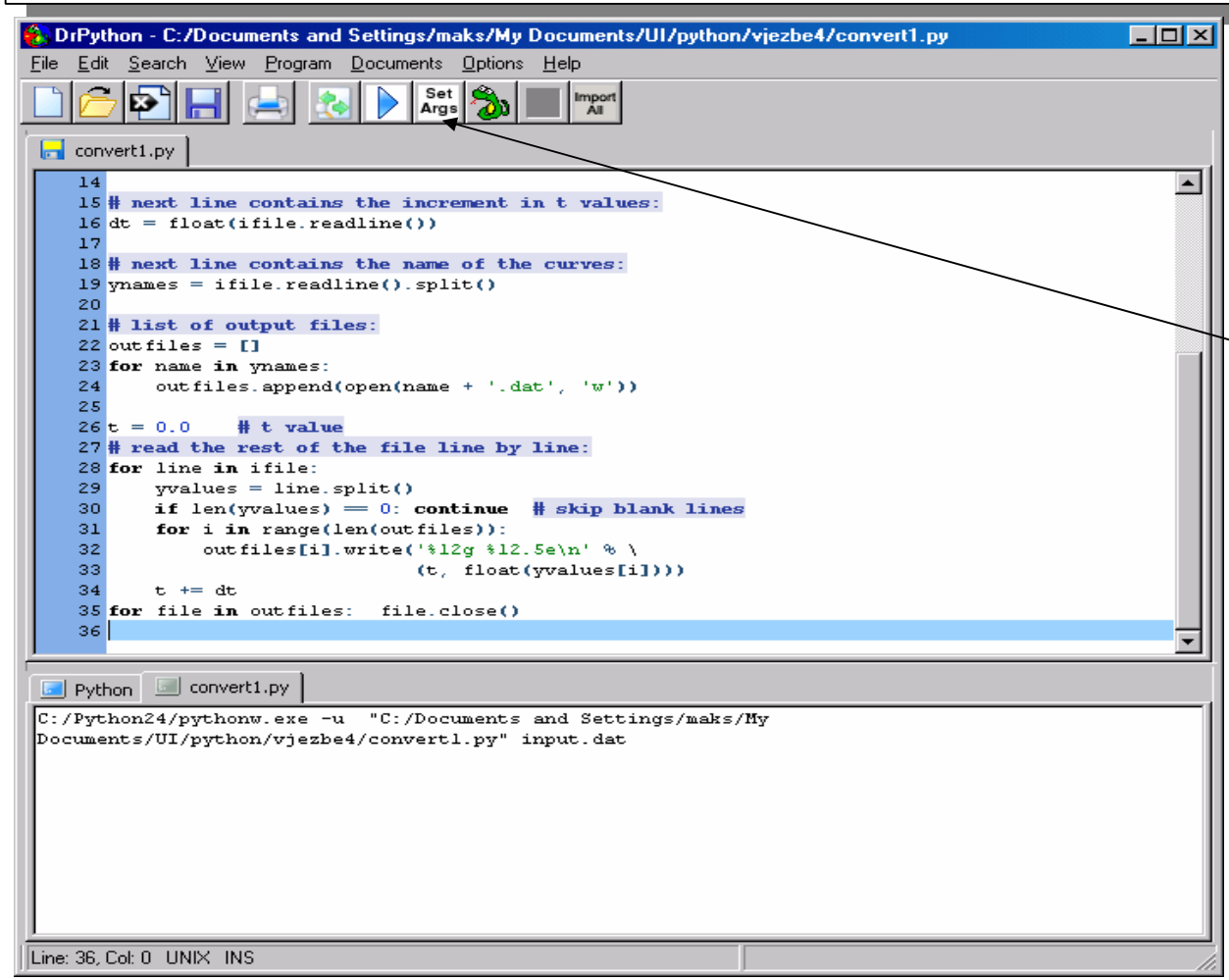
Tok programa:

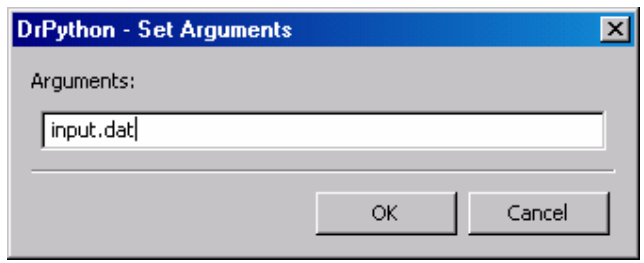
1. Otvori file, ime je prvi argument prilikom pozivanja programa. Ako fali argument obavijesti korisnika.
2. Pročitaj i preskoči prvu liniju, komentar.
3. Pročitaj vremenski korak iz druge linije
4. Pročitaj imena fileova pomoću trećeg reda. Napravi listu file objekata za različite fileove.
5. Pročitaj ostatak filea, liniju po liniju, razdvoji liniju u y vrijednosti i zapiši vrijednost u odgovarajući file zajedno s generiranim vremenom.

skripta

```
#!/usr/bin/env python
import sys, math, string
usage = 'Usage: %s infile' % sys.argv[0]
try:
        infilename = sys.argv[1]
except:
        print usage; sys.exit(1)
ifile = open(infilename, 'r') # open file for reading
# read first comment line (no further use of it here):
line = ifile.readline()
# next line contains the increment in t values:
dt = float(ifile.readline())
# next line contains the name of the curves:
ynames = ifile.readline().split()
# list of output files:
outfiles = []
for name in ynames:
        outfiles.append(open(name + '.dat', 'w'))
t = 0.0 # t value
# read the rest of the file line by line:
for line in ifile:
        yvalues = line.split()
        if len(yvalues) == 0: continue # skip blank
        for i in range(len(outfiles)):
                outfiles[i].write('%12g %12.5e\n'%\
                        (t, float(yvalues[i])))
        t += dt
for file in outfiles: file.close()
```

drpython i argumenti





skripta

Modifikacije na prvu verziju skripte.

Možemo cijeli file učitati u listu.

```
f = open(infilename, 'r'); lines = f.readlines(); f.close()
```

dt - vremenski korak dobivamo iz lines[1].

imena

petlja, brojevi

dictionary

```
# the third line contains the name of the time series:
ynames = lines[2].split()

# store y data in a dictionary of lists of floats:
y = {}            # declare empty dictionary
for name in ynames:
    y[name] = [] # empty list (of y values of a time series)

# load data from the rest of the lines:
for line in lines[3:]:
    yvalues = [float(x) for x in line.split()]
    if len(yvalues) == 0: continue  # skip blank lines
    i = 0  # counter for yvalues
    for name in ynames:
        y[name].append(yvalues[i]); i += 1
```

skripta

```
yvalues = [float(x) for x in line.split()]
```

Linija programa razdvaja liniju u listu stringova koje pretvara u realne brojeve primjenom funkcije float.
Zadnja petlja treba varijablu *i* koja broji o kojoj se vrijednosti radi (indeks) u listi *yyvalue*. Lijepša sintaksa je

```
for name, yvalue in zip(ynames, yvalues):
    y[name].append(yvalue)
```

gdje zip omogućuje iteriranje preko više varijabli.

zip

loop over a common index,

```
for i in range(len(xlist)):
    x = xlist[1];  y = ylist[i];  z = zlist[i]
    # or more compactly:  x, y, z = xlist[i], ylist[i], zlist[i]
    # work with x, y, and z
```

A shorter and more Pythonic alternative is to apply the `zip` function:

```
for x, y, z in zip(xlist, ylist, zlist):
    # work with x, y, and z
```

skripta

Na kraju skripte pišemo vrijednosti t i y u file

```
for name in y.keys():
    ofile = open(name+'.dat', 'w')
    for k in range(len(y[name])):
        ofile.write('%12g %12.5e\n' % (k*dt, y[name][k]))
    ofile.close()
```

convert1.py - skripta za prvu verziju

convert2.py - skripta nakon promjena

direktorij vjezbe4

Klase

- Kao kod modula, klase sadrže imena varijabli, tzv. namespace
  - nasljeđivanje
  - višestruke kopije klase
  - sadrže podatke o trenutnom stanju
  - više klasa možemo definirati u jednom file-u (modulu)
  - OOP - objektno orijentirano programiranje
  - *self* - pointer na klasu, kod definiranja metoda (funkcija) u klasi prvi argument mora biti *self*
    - self je ekvivalentan pointeru this u C++, self u Javi

Klase

Koristili smo primjer za modul:

```
# mymodule.py
""" This is module: mymodule
        mymodule prints Hi
"""
def sayhi():
        print "Hi, module speaking"

version=0.1
# end of mymodule
```

doc string

funkcija

varijabla

komentar

Učitamo modul

```
>>> import mymodule
>>> dir()
['__builtins__', '__doc__', '__name__', 'mymodule', 'os', 'sys']
>>> dir(mymodule)
['__builtins__', '__doc__', '__file__', '__name__', 'sayhi', 'version']
>>>
```

Klase

```
>>> mymodule.version
0.10000000000000001
>>> mymodule.sayhi()
Hi, module speaking
>>> sayhi()
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'sayhi' is not defined
>>>
```

Funkcije i varijable skrivene u modulu.

```
>>>
>>> from mymodule import *
>>> dir()
['__builtins__', '__doc__', '__name__', 'mymodule', 'os',
'sayhi', 'sys', 'version']
>>> version
0.10000000000000001
>>> sayhi()
Hi, module speaking
>>>
```

Funkcije i varijable sad se nalaze u globalnom prostoru.

Klase

```
#!/usr/bin/python
# Filename: method.py

class Person:
        def sayHi(self):
                print 'Hello, how are you?'

p = Person()
p.sayHi()

# This short example can also be written as Person().sayHi()
```

Definiramo klasu Person

funkcija (method)

p je klasa Person i sadrži funkciju sayHi

Output

```
$ python method.py
Hello, how are you?
```

Funkcija sayHi() nema parametara, medjutim self se nalazi u definiciji

Klase

Metoda __init__ se poziva prilikom stvaranja klase, pomoću nje postavljamo početne vrijednosti. Ekvivalentna je konstruktoru u C++.

```
#!/usr/bin/python
# Filename: class_init.py

class Person:
        def __init__(self, name):
                self.name = name
        def sayHi(self):
                print 'Hello, my name is', self.name

p = Person('Swaroop')
p.sayHi()

# This short example can also be written as Person('Swaroop').sayHi()
```

```
$ python class_init.py
Hello, my name is Swaroop
```

Klase

```
#!/usr/bin/python
# Filename: objvar.py
class Person:
        '''Represents a person.'''
        population = 0
def __init__(self, name):
        '''Initializes the person's data.'''
        self.name = name
        print '(Initializing %s)' % self.name
        # When this person is created, he/she
        # adds to the population
        Person.population += 1
```

```
def __del__(self):
        '''I am dying.'''
        print '%s says bye.' % self.name
        Person.population -= 1
        if Person.population == 0:
                print 'I am the last one.'
        else:
                print 'There are still %d people left.'
%               Person.population
```

Klase

```
def sayHi(self):
        '''Greeting by the person. Really, that's all it does.'''
        print 'Hi, my name is %s.' % self.name
def howMany(self):
        '''Prints the current population.'''
        if Person.population == 1:
                print 'I am the only person here.'
        else:
                print 'We have %d persons here.' %
Person.population
```

```
swaroop = Person('Swaroop')
swaroop.sayHi()
swaroop.howMany()
kalam = Person('Abdul Kalam')
kalam.sayHi()
kalam.howMany()
swaroop.sayHi()
swaroop.howMany()
```

```
$ python objvar.py
(Initializing Swaroop)
Hi, my name is Swaroop.
I am the only person here.
(Initializing Abdul Kalam)
We have 2 persons here.
Hi, my name is Swaroop.
We have 2 persons here.
Abdul Kalam says bye.
There are still 1 people left.
Swaroop says bye.
I am the last one.
```

prostor varijabli

```
!/usr/bin/python
# Filename: objvar1.py
class Person:
        '''Represents a person.'''
        print "Making a class ... "
        population = 0

        def __init__(self, name='John Doe'):
                '''Initializes the person's data.'''
                self.name=name
                print '(Initializing %s)' % self.name
                # When this person is created,
he/she
                # adds to the population
                Person.population += 1
```

modificirani program za klase

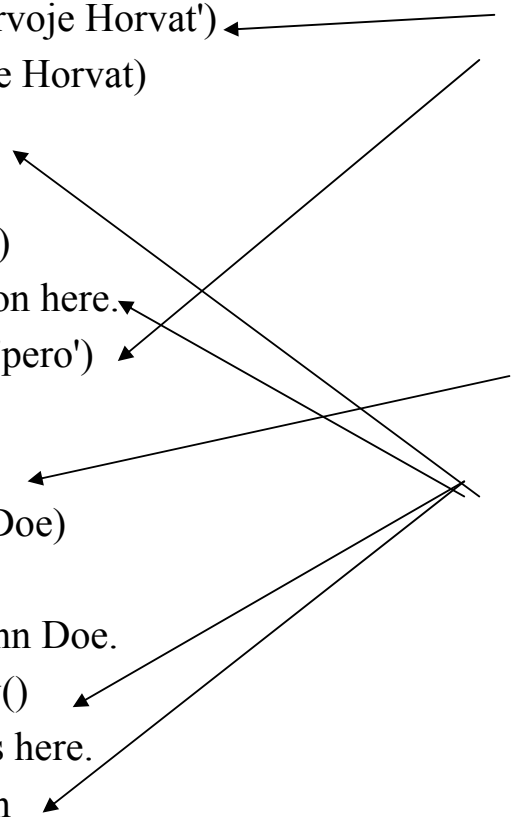
import - varijable inicijalizirane

Person.varijabla - globalna u prostoru klase

self.varijabla - svaka instanca ima svoju varijablu

Klase

```
>>> from objvar1 import *
Making a class ...
>>> hr=Person('Hrvoje Horvat')
(Initializing Hrvoje Horvat)
>>> hr.population
1
>>> hr.howMany()
I am the only person here.
>>> pero=Person('pero')
(Initializing pero)
>>> m1=Person()
(Initializing John Doe)
>>> m1.sayHi()
Hi, my name is John Doe.
>>> m1.howMany()
We have 3 persons here.
>>> m1.population
3
```



__init__ se poziva prilikom incijalizacije (stvaranja) klase, argumenti se prenose na funkciju.

Keyword ili Default argumenti u __init__ dobivamo klasu koja prepoznaje varijabilni broj argumenata (ali ne proizvoljno).

Varijable KLASE dijele svi objekti (instance) te klase. Postoji samo jedna kopija KLASE, kad neki objekt promjeni varijablu u KLASI, svi objekti vide istu vrijednost.

Klase

- Sve varijable klase su *public,* a funkcije *virtualne*
- Konvencija:
  - sve varijable ili funkcije koje će ce koristiti samo unutar klase trebaju početi znakom _
  - sve druge varijable/funkcije se mogu koristiti u drugim objektima/klasama
  - __ su privatne funkcije
  - __del__ je analogan destruktoru u C++
    - poziva se prilikom destrukcije objekta (vidi objvar.py primjer)

Nasljeđivanje

- Nova klasa može naslijediti varijable i metode
  - Napiše se jednostavna klasa baze koja sadrži postavljanje vrijednosti
  - Nasljeđujemo je prilikom definiranja nove klase, argument je ime klase koju nasljeđujemo
  - __init__ klase baze se mora eksplicitno pozvati pomoću self varijable
  - nasljeđivanje više klase se zove višestruko nasljeđivanje (eng. multiple inheritance)

Nasljeđivanje

```
class SchoolMember:
        '''Represents any school member.'''
        def __init__(self, name, age):
                self.name = name
                self.age = age
                print '(Initialized SchoolMember: %s)' % self.name

        def tell(self):
                '''Tell my details.'''
                print 'Name:"%s" Age:"%s"' % (self.name, self.age),
class Teacher(SchoolMember):
        '''Represents a teacher.'''
        def __init__(self, name, age, salary):
                SchoolMember.__init__(self, name, age)
                self.salary = salary
                print '(Initialized Teacher: %s)' % self.name

        def tell(self):
                SchoolMember.tell(self)
                print 'Salary: "%d"' % self.salary
```

Nasljeđivanje

```
class Student(SchoolMember):
        '''Represents a student.'''
        def __init__(self, name, age, marks):
                SchoolMember.__init__(self, name, age)
                self.marks = marks
                print '(Initialized Student: %s)' % self.name

        def tell(self):
                SchoolMember.tell(self)
                print 'Marks: "%d"' % self.marks

t = Teacher('Mrs. Shrividya', 40, 30000)
s = Student('Swaroop', 22, 75)

print # prints a blank line

members = [t, s]
for member in members:
        member.tell() # works for both Teachers and Students
```

Nasljeđivanje

```
$ python inherit.py
(Initialized SchoolMember: Mrs. Shrividya)
(Initialized Teacher: Mrs. Shrividya)
(Initialized SchoolMember: Swaroop)
(Initialized Student: Swaroop)

Name:"Mrs. Shrividya" Age:"40" Salary: "30000"
Name:"Swaroop" Age:"22" Marks: "75"
```

Klase

Višestruko nasljeđivanje

```
class A:
    def set(self, a):
        self.a = a;  print 'A.set'

class B:
    def set(self, b):
        self.b = b;  print 'B.set'

class C(A, B):
    def set(self, c):
        self.c = c;  print 'C.set'

    def somefunc(self, x, y):
        A.set(self, x)    # call base class method
        B.set(self, y)    # call base class method
        self.set(0)       # call C's set method
```

Primjena:

```
>>> c = C()
>>> c.somefunc(2,3)
A.set
B.set
C.set
>>> print c.__dict__
{'a': 2, 'c': 0, 'b': 3}
```

Integriranje

Trapezoidal rule: $\int_{-1}^{1} f(x)dx \approx f(-1)+f(1)$ općenito napisano $\int_{-1}^{1} f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i)$

gdje su w_i x_i težine i točke integracije.

```
class Trapezoidal:
    """The Trapezoidal rule for integrals on [-1,1]."""

    def __init__(self):
        self.setup()

    def setup(self):
        self.points = (-1, 1)
        self.weights = (1, 1)

    def eval(self, f):
        sum = 0.0
        for i in range(len(self.points)):
            sum += self.weights[i]*f(self.points[i])
        return sum
```

```
# usage:
rule = Trapezoidal()
integral = rule.eval(lambda x: x**3)
```

Integriranje

Simpson's rule,

$$\int_{-1}^{1} f(x)dx \approx \frac{1}{3}f(-1) + \frac{4}{3}f(0) + \frac{1}{3}f(1),$$

Gauss-Legendre rule,

$$\int_{-1}^{1} f(x)dx \approx f(-\frac{1}{\sqrt{3}}) + f(\frac{1}{\sqrt{3}}).$$

```
class Integrate:
    def __init__(self):
        self.setup()

    def setup(self):
        # to be overridden in subclasses:
        self.weights = None
        self.points = None

    def eval(self, f):
        sum = 0.0
        for i in range(len(self.points)):
            sum += self.weights[i]*f(self.pc
        return sum
```

```
class Trapezoidal(Integrate):
    def setup(self):
        self.points = (-1, 1)
        self.weights = (1, 1)

class Simpson(Integrate):
    def setup(self):
        self.points = (-1, 0, 1)
        self.weights = (1/3.0, 4/3.0, 1/3.0)

class GaussLegendre2(Integrate):
    def setup(self):
        p = 1/math.sqrt(3)
        self.points = (-p, p)
        self.weights = (1, 1)
```

Integriranje

$$\int_a^b f(x)dx = \sum_{j=1}^{n} \int_{\Omega j} f(x)dx,$$

$$\Omega_j = [(j-1)h, jh], \quad h = \frac{b-a}{n},$$

$$\int_{\Omega_j} f(x)dx = \int_{-1}^{1} g(\xi)\frac{h}{2}d\xi, \quad g(\xi) = f(x(\xi)),\ x(\xi) = (j - \frac{1}{2})h + \frac{h}{2}\xi\,.$$

```
def f(x):
    print 'f(%g)=%g' % (x, 2*x)
    return 2*x
```

test funkcija u integrate.py

Integriranje

```
def integrate(integrator, a, b, f, n):
    # integrator is an instance of a subclass of Integrator
    sum = 0.0
    h = (b-a)/float(n)
    g = TransFunc(f, h)
    for j in range(1, n+1):
        g.j = j
        sum += integrator.eval(g)
    return sum

 class TransFunc:
     def __init__(self, f, h):
         self.f = f;  self.h = h

     def coor_mapping(xi):
         """Map local xi in (-1,1) in interval j to global x."""
         return (self.j-0.5)*self.h + 0.5*self.h*xi

     def __call__(self, xi):
         x = self.coor_mapping(xi)
         return self.f(x)
```

g varijabla je funkcija $g(\xi)$

Integriranje

Primjer u integrate.py:

```
if __name__ == '__main__':
    import sys, math
    try:
        classname = sys.argv[1]
    except:
        print 'Usage: %s Simpson|Trapezoidal|GaussLegendre2' % sys.argv[0]
        sys.exit(1)
    s = eval(classname + '()')
    v = s.eval(lambda x: 2*x)
    print 'integral of f(x)=2*x from -1 to 1 (=0):', v
    v = integrate(s, 0, 2, f, 2)
    print 'integral of f(x)=2*x from 0 to 2 (=4):', v
```

npr. s=Trapezoidal()

v=Trapezoidal(2*x)

općenita metoda

Integriranje

C:/Python24/pythonw.exe -u "C:/Documents and Settings/maks/My Documents/UI/python/scripting/src/py/examples/integrate.py"
Usage: C:/Documents and Settings/maks/My Documents/UI/python/scripting/src/py/examples/integrate.py Simpson|Trapezoidal|GaussLegendre2

Bez argumenata, i argument je Trapezoidal

C:/Python24/pythonw.exe -u "C:/Documents and Settings/maks/My Documents/UI/python/scripting/src/py/examples/integrate.py" Trapezoidal
integral of f(x)=2*x from -1 to 1 (=0): 0.0
f(0)=0
f(1)=2
f(1)=2
f(2)=4
integral of f(x)=2*x from 0 to 2 (=4): 4.0

Integriranje

Interaktivno koristimo integrate.py u drPythonu

```
>>> from integrate import *
>>> v1=Trapezoidal()
>>> v1.eval(lambda x: 2*x)
0.0
>>> v1.eval(f)
f(-1)=-2
f(1)=2
0.0
```

lambda funkcija

funkcija f definirana u integrate.py

Literatura

- A Byte of Python, Swaroop C H
- Python - Osnove, M. Essert i I. Vazler, Osijek 2007.
- NUMERICAL METHODS IN ENGINEERING WITH Python, Jaan Kiusalaas, Cambridge University Press
- Python cookbook, O'Reilly
- Python Scripting for Computational Science, Hans Petter Langtangen, Springer-Verlag
- *Programming Python*, Mark Lutz, O'Reilly
- *Learning Python*, Mark Lutz, David Ascher, Frank Wilson, O'Reilly

Literatura

- *Python Programming on Win32*, Mark Hammond and Andy Robinson, O'Reilly
- *Python Standard Library*, Fredrik Lundh,O'Reilly
- *Python in a Nutshell*, by Alex Martelli, O'Reilly
- *Python & XML*, by Christopher A. Jones and Fred L. Drake, Jr. , O'Reilly
- *Python Web Programming*, by Steve Holden, New Riders