Python logo

Izbrana poglavja iz informatike

Uvod v Python

Vladimir Batagelj
Univerza v Ljubljani
FMF, matematika

različica: 29. marec 2009 / 00 : 19

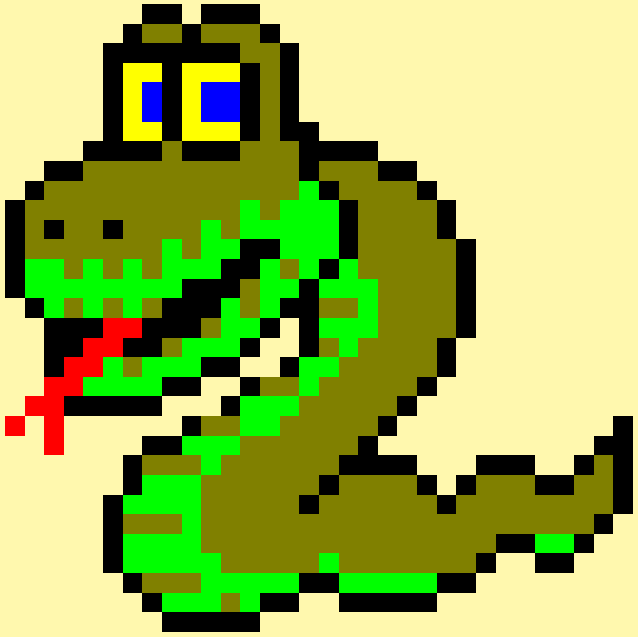
Kazalo

Python

Guido van Rossum

Programski jezik Python je konec osemdesetih let zasnoval Guido van Rossum kot naslednika jezika ABC. Ime je dobil po BBC-jevi nadaljevanki ”Monty Python’s Flying Circus”. Guido van Rossum ima še danes glavno besedo pri razvoju Pythona – doživljenski prijazni samodržec (Benevolent Dictator for Life, BDFL).

Trenutno (konec 2008) je najbolje podprta različica 2.5. Izdana je tudi že različica 2.6, ki predstavlja povezavo z različico 3.0 (izdana 3. decembra 2008). Ta prinaša rez v razvoju Pythona – vrsto novih sestavin in opustitev nekaterih neustreznih. Mi bomo uporabljali kar različico 3.0.



Python

. . . Python

Python temelji na tolmačenju prepreletenim s prevajanjem v strojni jezik za Pythonov navidezni stroj. Tolmač najdemo na Pythonovem spletišču.

Python je predmetni (objektni) jezik, ki združuje ukazni in funkcijski pristop. Ima preprosto slovnico in je varčen s sestavinami. Kljub temu je po zmogljivostih primerljiv z drugimi splošno namenskimi jeziki (fortran, basic, lisp, pascal, ada, C, C++, C#, java, scheme, ML, ruby, . . .).

Posebno moč mu dajejo obsežne knjižnice za posebne naloge (grafika, slikovni vmesnik, podatkovne baze, delo s spletom, podpora XML, jezikovne analize, . . .).

Osnovni Python razvija Python Software Foundation. Precej razširjena je tudi izvedba Pythona podjetja ActiveState. IronPython.

Python kot računalo

Ko namestimo Python ga je najenostavneje uporabiti tako, da poženemo njegov tolmač IDLE. Pojavi se tolmačevo okno z nekaj vrsticami napisa in kot zadnja vrstica tolmačev pozivnik >>>. Začnemo lahko vnašati različne izraze:

```
>>> 3 + 4
7
>>> a = 3+4
>>> a
7
>>> b=(a+3)*2
>>> b
20
>>> a, b = b, a
>>> a, b
(20, 7)
>>> A = 10
>>> a, A
(20, 10)
>>> a = b = 0
>>> a, b
(0, 0)
```

Python loči velike in male črke. Po že vnešenih vrsticah se lahko sprehajamo z `Alt-P` in `Alt-N`.

Python – knjižnice

```
>>> sin(0.5)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in ?
    sin(0.5)
NameError: name 'sin' is not defined

>>> import math
>>> math.sin(0.5)
0.47942553860420301
>>> from math import sin
>>> sin(0.5)
0.47942553860420301
>>> pow(2,100)
1267650600228229401496703205376
```

Python podpira poljubno velika cela števila.

Imena spremenljivk v Pythonu so sestavljena iz črk (Unicode), števk in znaka `'_'`. Prepovedana so rezervirana imena: `and, as, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while, with, yield, None`. Imena, ki se začenjajo z `'_'`, imajo v Pythonu posebno vlogo. V pogovornem načinu dela ime `'_'` označuje zadnjo vrednost.

Podatki v Pythonu

Števila

Python pozna cela števila **int**, dolga cela števila **long**, realna števila **float** in kompleksna števila **complex**. O zvrsti podatka povprašamo s funkcijo type.

```
 123, 3.14, 3.2e-12
 12345678901234567, 3+4j
 0177, 0x3afb
```

```
>>> 0177, 0x3afb
(127, 15099)
>>> pow(3+4j,5)
(-237-3116j)
>>> type(2**100)
<class 'int'>
```

`e=E, l=L, j=J, x=X`

Osmiške številke začnejo z ničlo `0`, šestnajstiške pa z `0x`. V Pythonu 3.0 osmiške številke začenjajo z `0o`, dvojiške pa z `0b`. Nova je tudi knjižnica `fractions`, ki prinaša podporo za računanje z ulomki – zvrst **Fraction**.

Številske operacije

Python pozna naslednje dvomestne številske operacije: `+` (seštevanje), `-` (odštevanje), `*` (množenje), `/` (pravo deljenje), `//` (celoštevilsko deljenje), `%` (ostanek pri celoštevilskem deljenju), `**` (potenciranje); operaciji min in max uporabljamo kot funkciji (s poljubnim številom argumentov).

Prepovedano je deljenje z 0. V Pythonu 3.0 neceloštevilske potence negativnih števil dajo kompleksen rezultat. Povsod pa deluje `complex(x)**y`. Za operacijo `**` velja računanje z desne proti levi.

Z enomestno operacijo `-x` zamenjamo predznak števila x. S funkcijo `abs(`x`)` pa določimo njegovo absolutno vrednost.

Pri računanju Python po potrebi sam prilagaja zvrsti členov dvomestne operacije – prevlada širša zvrst, ki je tudi zvrst rezultata. Lahko pa pretvorbe tudi sami zahtevamo s funkcijami `int`, `long`, `float` in `complex`.

Za pretvorbo realnega števila v najbližjo celo vrednost je na voljo funkcija `round`. Pozor, vrne realno število.

Številske funkcije

Večino običajnih številskih funkcij najdemo v knjižnici math ali `dir(math)` ali `help(math)`: `ceil, floor, exp, log, log10, pow, sqrt, sin, cos, tan, asin, acos, atan, atan2, sinh, asinh, degrees, radians, ...`

V knjižnici math sta tudi konstanti π in e:

```
>>> from math import *
>>> pi
3.141592653589793
>>> e
2.718281828459045
```

Python pozna tudi tročleno operacijo a `if` p `else` b

```
>>> b=True
>>> 5 if b else 3
5
>>> b=False
>>> 5 if b else 3
3
```

Izračun je kratkostični.

Logične vrednosti

V Pythonu sta logični vrednosti `True` in `False` predstavljeni s številoma `1` in `0`. Nasplošno pa imajo ničelni/prazni podatki vlogo `False`; ostali `True`.

`x or y`, `x and y` (le delni – kratkostični izračun, če je izid znan)

```
>>> 3 or 4, 5 and 6, 0 or 3, 5 and 0
(3, 6, 3, 0)
```

`not` x, `any` $(l_1, l_2, \ldots, l_k)$, `all` $(l_1, l_2, \ldots, l_k)$

```
>>> not not 5
True
```

```
<, <=, >, >=, ==, <>, !=, is, is not, in, not in
x | y, x ^ y, x & y    (po bitih or, xor, and)
x << n, x >> n         (pomik)
```

```
>>> 1 << 5
32
>>> 3 < 4 < 5
True
```

```
a op= b      okrajšava za      a = a op b
+=, -=, *=, /=, //=, **=, %=, &=, |=, ^=, >>=, <<=
```

Nizi

```
"dober dan",'kokoš',''

>>> print('koko"s')
koko"s

>>> m = 'Ljubljana'
>>> 'a' in m
True
>>> z = "a" "b" + "c"
>>> z
'abc'
>>> z = 3        (spremenljivke so v Pythonu 'kazalci',
>>> z             zvrst pripada podatkom)
3
>>> m*3
'LjubljanaLjubljanaLjubljana'
>>> len(m)
9
>>> m[4]
'l'
>>> m[3:5]
'bl'
>>> m[:4]
'Ljub'
>>> m[:-1]
'Ljubljan'
>>> m[-1], m[-2]
('a', 'n')
```

. . . nizi

```
>>> napis = """Dober
dan
vsem skupaj"""
>>> napis
'Dober\ndan\nvsem skupaj'
>>> print(napis)
Dober
dan
vsem skupaj

\n, \r, \v, \t, \f, \0XY, \xXY, \000
\a (bell), \b (backspace), \e (escape)
```

V starejših različicah Pythona je bil `print` stavek, v Pythonu 3.0 pa je postal funkcija.

Branje podatkov

```
>>> c = input('c = ')
c = 14
>>> c
'14'
>>> int(c)
14
>>> a = eval(input('a = '))
a = 3
>>> b = eval(input('b = '))
b = 4
>>> print(a,'+',b,'=',a+b)
3 + 4 = 7
>>> print(a,'+',b,'=',a+b,sep='')
3+4=7
```

Funkcija `input(`*poziv*`)` izpiše *poziv*ni niz in prebere vnešeni podatek kot niz znakov. S funkcijo `eval` izračunamo njegovo vrednost – dobimo število.

V funkciji `print` smo uporabili parameter `sep`, ki določa niz, ki se doda med izpisanimi členi. Funkcija `print` običajno zaključi vrstico – na koncu izpiše znak `'\n'`. Zaključni niz lahko določimo s parametrom `end`.

Programski način dela

V IDLE izberemo `File/New window` in v novo okno vnesemo zaporedje stavkov

```
a = eval(input('a = '))
b = eval(input('b = '))
print(a,'+',b,'=',a+b,sep='')
```

Shranimo ga z `File/Save as` na datoteko `vsota.py`. Nato izberemo `Run/Run module`. Dobimo:

```
>>> ================================ RESTART ================
>>>
a = 7
b = 5
7+5=12
>>>
```


Osnovni krmilni stavki

Stavek nadaljujemo v novo vrsto z \ Tudi vsebina [ ] se lahko razteza čez več vrstic. Več stavkov v vrstici ločimo s ;

```
>>> a = 3; b = 4
>>> a
3
```

označuje vrstično pojasnilo

. . . krmilni stavki

Podrejene stavke določimo z zamikanjem.

```
if p1 :           # obvezno zamikanje
   stavki1
elif p2 :
   stavki2
elif p3 :
   stavki3
else:
   stavki

while p :
   stavki1
else:
   stavki2

break             # zapusti zanko
continue          # na začetek zanke
pass              # prazni stavek
```

deli `else` oziroma `elif` niso obvezni.

. . . krmilni stavki

Dopolnimo naš program, tako da bo sešteval vnešene pare števil vse dokler ne vnesemo kot prvi člen 0 :

```
while True:
   a = eval(input('a = '))
   if a == 0 :
      print('nasvidenje')
      break
   b = eval(input('b = '))
   print(a,'+',b,'=',a+b,sep='')
```

Shranimo na `vsota2.py` in poženemo:

```
>>> ================================ RESTART ============
>>>
a = 312
b = 35
312+35=347
a = -103
b = 98
-103+98=-5
a = 0
nasvidenje
>>>
```

. . . krmilni stavki

Dodamo štetje korakov k, shranimo na `vsota3.py` in poženemo:

```
# -*- coding: utf-8 -*-
k = 0
while k < 3:
   k = k+1
   a = eval(input('a = '))
   if a == 0 :
      break
   b = eval(input('b = '))
   print(k,'. račun: ',a,'+',b,'=',a+b,sep='')
else:
   print('dovolj je bilo')
print('nasvidenje')
```

```
>>> ================================ RESTART =============
>>>                                  >>>
a = 3                                a = 11
b = 2                                b = 27
1. račun: 3+2=5                      1. račun: 11+27=38
a = 4                                a = -33
b = 7                                b = 47
2. račun: 4+7=11                     2. račun: -33+47=14
a = 5                                a = 0
b = 5                                nasvidenje
3. račun: 5+5=10                     >>>
dovolj je bilo
nasvidenje
>>>
```

Izjeme

V Pythonu prestrežemo napake pri izvajanju skupine stavkov tako, da jih oklenemo s stavkom try

```
try:
   stavki
except izjema1:
   obdelava1
...
except izjema2:
   obdelavak
else:
   obdelava
finally:
   počisti
```

Stavki `except` poskrbijo za posamezne vrste napak; če ni najdena zanjo poskrbi stavek `else`. Stavek `finally` se se vselej izvrši – tudi, če nastopijo napake v obdelavi izjeme.

Izjemo lahko tudi sami sprožimo s stavkom

`raise` izjema, sporočilo.

Ustvarimo lahko tudi lastne izjeme. podrobno.

. . . izjeme

Stavek

`assert` pogoj, sporočilo

omogoča preverjati ali so izpolnjene predpostavke. Če pogoj ni izpolnjen, se sproži izjema. Njegovo delovanje nadzira sistemska spremenljivka __debug__.

```
def koren(x):
  assert x >= 0, "koren - negativen argument"
  return x**0.5
```

Objekti

Python je *objektni* (predmetni) jezik. Objekti pripadajo različnim zvrstem ali *razredom*. Za objekte posamezne zvrsti sta značilna nabor *lastnosti* (podatki) in nabor *metod* (ukazi, funkcije, operacije).

Lastnosti določajo stanje objekta, metode pa kaj lahko z njim počnemo. Metode omogočajo *doseganje* (poizvedovanje o) lastnosti(h) oziroma *spreminjanje* stanja objekta. Objekti posameznega razreda so lahko ali *spremenljivi* (njihovo stanje se lahko spreminja) ali pa *pribiti* (stalni, nespremenljivi).

Do lastnosti $last$ objekta obj pridemo z imenom `obj.last`.

Uporabo metode met na objektu obj pa zahtevamo z izrazom `obj.met(par)`, kjer so par (morebitni) parametri metode. Oklepaje je potrebno napisati tudi kadar metoda nima parametrov.

Nekaj razredov je v Python že vgrajenih. Kasneje bomo spoznali tudi, kako ustvarimo svoje lastne razrede. Poseben objekt `None` označuje, da vrednost objekta (še) ni določena.

Zaporedja

Python pozna nekaj vrst *zaporedij*: ponovnike, nize, sezname in nabore.

Nad vsemi zaporedji sta definirani funkciji min, max in len. $len(s)$ vrne dolžino (število členov) zaporedja s. Operacija $+$ določa stikanje zaporedij, izraz $s * n$ oziroma $n * s$ pa določa stik n izvodov zaporedja s. Uporabi istega zapisa za različne zvrsti objektov pravimo *večličnost* (polimorfizem).

Izraz a `in` s preverja ali objekt a nastopa v zaporedju s. Pri nizih preverja tudi, ali je niz a podniz niza s.

i-ti člen zaporedja s dosegamo z izrazom `s[i]`. V Pythonu štejemo od 0 naprej. Z negativnimi indeksi $s[-i]$ dosegamo člene z zadnjega konca.

Z izrazom `s[i:j]` določimo *izrez* iz zaporedja s, ki ga sestavljajo členi od i-tega (vključno) do j-tega (izključno). Izraz `s[i:j:k]` določa izrez, pri čemer vzamemo le vsak k-ti člen.

Za delo z zaporedji je na voljo poseben krmilni stavek `for` a `in` s : $stavki$, ki izvaja $stavke$ zaporedoma za vsak člen a zaporedja s.

Še o nizih

Nizi, zvrst **str**, so pribiti podatki. Za delo z nizi je na voljo veliko metod: `strip`, `rstrip`, `lstrip`, `upper`, `lower`, `split`, `join`, `count`, `find`, ...Za podrobnosti glejte še `help(str)` in metode v knjižnici `string`.

V Pythonu 3.0 so nizi (zvrst **str**)unicodski, navadni nizi (ASCII) pa so obravnavani kot zaporedja (8-bitnih) zlogov (kod) – zvrsti **bytes** in **bytearray**.

Posamezne znake, ki niso na tipkovnici, vnesemo z njihovo unicodsko kodo `\u`$XYZW$, kjer je $XYZW$ šestnajstiško zapisana koda znaka.

Č – `\u010C`, č – `\u010D`, Š – `\u0160`, š – `\u0161`,
Ž – `\u017D`, ž – `\u017E` .

```
"\u017Eari\u0161\u010De"
```

Knjižnica za delo z regularnimi izrazi `re`.

Funkcije: `chr`, `ord`, `str`, `oct`, `hex`, `repr`, `eval`.

Seznami

Seznami so spremenljivi podatki.

```
[], [ 'b', 'bcd, 3, [ ['x', 1], '3+4' ], a, 7.5 ]

>>> a = [ 'Nova', 'Gorica' ]
>>> b = [ 'b', 'bcd', 3, [ ['x', 1], '3+4' ], a, 7.5 ]
>>> b
['b', 'bcd', 3, [['x', 1], '3+4'], ['Nova', 'Gorica'], 7.5]
>>> b[4], b[3], b[-1]
(['Nova', 'Gorica'], [['x', 1], '3+4'], 7.5)
>>> b[1:3]
['bcd', 3]
>>> len(b)
6
>>> 'bcd' in b, 'x' in b
(1, 0)
```

Metode: `list`, `append`, `extend`, `count`, `index`, `insert`, `pop`, `remove`, `reverse`, `sort`, ... (podrobneje list). `range`.

Seznami – operacije

```
L.append(X), L.sort(), L.index(X), L.reverse()
del L[i:j]

>>> a.reverse()
>>> b
['b', 'bcd', 3, [['x', 1], '3+4'], ['Gorica', 'Nova'], 7.5]
>>> b.index(7.5), b.index(a)
(5, 4)
>>> del b[3:5]
>>> b
['b', 'bcd', 3, 7.5]
>>> b.sort()
>>> b
[3, 7.5, 'b', 'bcd']

>>> a = [1, [2,3]]
>>> b = a
>>> c = [1, [2,3]]
>>> a == b, a == c, a is b, a is c
(1, 1, 1, 0)

>>> range(4)
[0, 1, 2, 3]
>>> range(5,11)
[5, 6, 7, 8, 9, 10]
>>> range(7,20,3)
[7, 10, 13, 16, 19]
```

Seznamski izrazi

Seznamski izraz (list comprehension)

`[` izraz `for` ime `in` zaporedje `if` pogoj `]`

Če oglate oklepaje zamenjamo z okroglimi, dobimo *ponovniški izraz*

`(` izraz `for` ime `in` zaporedje `if` pogoj `)`

ki ustvari ponovnik (generator).

```
>>> [ (x,x**2) for x in range(10) ]
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25), (6, 36),
 (7, 49), (8, 64), (9, 81)]
>>> ( (x,x**2) for x in range(10) )
<generator object at 0x00CB34E0>
>>> q = ( (x,x**2) for x in range(10) )
>>> q.next()
(0, 0)
>>> q.next()
(1, 1)
>>> q.next()
(2, 4)
>>> [ (x,y) for x in "abc" for y in range(1,4) ]
[('a', 1), ('a', 2), ('a', 3), ('b', 1), ('b', 2), ('b', 3),
 ('c', 1), ('c', 2), ('c', 3)]
```


Slovarji

```
{} prazen

>>> S = { 'tomo': 'tomaz.pisanski@fmf.uni-lj.si',
  'vlado': 'vladimir.batagelj@uni-lj.si',
  'andrej': 'andrej.mrvar@uni-lj.si' }
>>> S
{'andrej': 'andrej.mrvar@uni-lj.si', 'vlado': 'vladimir.batagelj@uni-lj.si',
 'tomo': 'tomaz.pisanski@fmf.uni-lj.si'}
>>> S['vlado']
'vladimir.batagelj@uni-lj.si'
>>> S.keys()
['andrej', 'vlado', 'tomo']
>>> S.values()
['andrej.mrvar@uni-lj.si', 'vladimir.batagelj@uni-lj.si',
 'tomaz.pisanski@fmf.uni-lj.si']
>>> S['matjaz'] = 'matjaz.zaversnik@fmf.uni-lj.si'
>>> S.has_key('matija'), S.has_key('matjaz')
(0, 1)
>>> S['vlado'] = 'vladimir.batagelj@fmf.uni-lj.si'
>>> S['vlado']
'vladimir.batagelj@fmf.uni-lj.si'
>>> len(S)
4
>>> del S['vlado']
>>> S.has_key('vlado')
0
>>> S[3] = ['a', 395, {1: 'x', 5: 'w'}]
>>> S[3][2][5]
'w'
```

`for` *key* `in` *S*: *stavki*

Nabori

So podobni seznamom, le da ne dopuščajo operacij na mestu – so pribiti.

Uporabljamo jih, kadar želimo biti gotovi, da se vrednost ne spreminja.

```
>>> a = ( 'a', 'b', 'c', 3, 4, [0, 1], 2004)
>>> a
('a', 'b', 'c', 3, 4, [0, 1], 2004)
>>> a[5]
[0, 1]
>>> len(a)
7
>>> a[2]
'c'
>>> a[2] = 'z'
Traceback (most recent call last):
  File "<pyshell#92>", line 1, in ?
    a[2] = 'z'
TypeError: object doesn't support item assignment
>>> a = a[:2] + ('z',) + a[3:]
>>> a
('a', 'b', 'z', 3, 4, [0, 1], 2004)

>>> zip(range(1,6),list('abcde'))
[(1, 'a'), (2, 'b'), (3, 'c'), (4, 'd'), (5, 'e')]
```

Z izrazi oblike *predloga* `%` *nabor* lahko oblikujemo izpis (glej določila).

```
>>> a=324; x=355./113; i='petek'
>>> "n=%8i  P=%9.6f  d=%e %10s" % (a,x,math.pi-x,i)
'n=     324  P= 3.141593  d=-2.667642e-007      petek'
```

Množice

Množice so neurejene zbirke. Ne upoštevajo se tudi večkratne pojavitve podatkov. Python pozna dve vrste množic – spremenljive `set` in pribite `frozenset`.

```
>>> set('ljubljana')
set(['a', 'b', 'j', 'l', 'n', 'u'])
>>> frozenset('ljubelj')
frozenset(['e', 'b', 'j', 'u', 'l'])
```

x `in` s, x `not in` s, `len`,

`issubset`, `issuperset`, `union`, `intersection`, `difference`, `symmetric_difference`, `copy`

`s <= t , s >= t, s | t, s & t, s - t, s ^ t`

Podrobneje set

. . . množice

```
>>> a=set('ljubljana'); b=set('ljubelj')
>>> a|b
set(['a', 'b', 'e', 'j', 'l', 'n', 'u'])
>>> a&b
set(['b', 'u', 'l', 'j'])
>>> a^b
set(['a', 'e', 'n'])
>>> a-b
set(['a', 'n'])
```

Na spremenljivih množicah `set` so na voljo še metode: `update`, `intersection_update`, `difference_update`, `symmetric_difference_update`, `add`, `remove`, `discard`, `pop`, `clear`

`s |= t, s &= t, s -= t, s ^= t`

Tabele

Python spočetka ni poznal običajnih tabel. Za večino uporab tabel sta primerni zvrsti seznam (`list`) in slovar (`dict`). Prave tabele je omogočila šele knjižnica array, ki pa ponuja le najnujnejše. Za zahtevnejše računske obdelave sta na voljo knjižnici NumPy in SciPy.

Prednostni vrstni red operacij sledi pravilom iz matematike.

Funkcije

```
def ime(p1,p2,...,pn):
   global v1, v2, ...,vk
   stavki
   return vrednost

def gcd(m,n):
  if n == 0: return abs(m)
  else: return gcd(n, m % n)

Imena - pravilo  LEGB   Local, Enclosing, Global, Built-in

return lahko vrne tudi nabor
   return v1, v2, v3
```

V p_i lahko uporabimo tudi obliko `p=v` kjer je `v` privzeta vrednost. To lahko uporabimo tudi pri klicu.

`*name` nabor prestalih mestnih dejanskih argumentov

`**name` slovar prestalih imenovanih dejanskih argumentov

Funkcije – lambda

```
lambda p1, p2, ..., pn: izraz
```

```
>>> f = lambda x, y : x*x + y*y
>>> f(3,4)
25
>>> (lambda x, y : x*x + y*y)(3,4)
25
>>> apply(f,(3,4))
25

>>> map((lambda x: x*x + x + 41),range(41))
[41, 43, 47, 53, 61, 71, 83, 97, 113, 131, 151, 173,
197, 223, 251, 281, 313, 347, 383, 421, 461, 503, 547,
593, 641, 691, 743, 797, 853, 911, 971, 1033, 1097,
1163, 1231, 1301, 1373, 1447, 1523, 1601, 1681]
```

Knjižnica `operator`: `add, mul, or_, and_, ...`

Ponovniki

Ponovniki (generatorji) so objekti, ki ob zahtevi vrnejo naslednji člen zaporedja. Ustvarimo jih lahko tudi kot funkcije, v katerih namesto stavka `return` uporabimo stavek `yield`. Ta vrne tekočo vrednost, a funkcije ne zapusti – pri naslednji zahtevi nadaljuje izvajanje za stvakom `yield`.

```
>>> def kva():
        for k in xrange(10):
            yield k**2
>>> for i in kva():
        print(i,end=" ")
0 1 4 9 16 25 36 49 64 81
>>> list(kva())
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> p = kva()
>>> p
<generator object at 0x00CBCD00>
>>> p.next(), p.next(), p.next(), p.next(), p.next()
(0, 1, 4, 9, 16)
```

Datoteke

Datoteka je zaporedje bitov shranjeno na pomožnem pomnilniku. Znakovne in dvojiške datoteke. Zaporedne in naključne.

Datoteko moramo najprej odpreti

d = `open`(datoteka, določila)

Osnovna določila so `'r'` (read), `'w'` (write), `'a'` (append). Če izbranemu določilu dodamo `'+'`, lahko z datoteke beremo in nanjo pišemo. Dodatni določili sta še `'b'` (binary) in `'t'` (text). Določilo `'U'` določa obravnavo `'\n'`, `'\r'` in `'\r\n'` kot konec vrstice.

Za delo z datotekami je na voljo več metod: read, readline, readlines, write, writelines, close, flush, seek, ...

```
for vrsta in open('podatki.txt','rU'):
   obdelaj(vrsta)
```

Python ponuja še veliko drugih možnosti za delo z datotekami.

Podatki so lahko program

Velika moč jezikov, ki temeljijo na tolmačenju je, da je meja med podatki in programom prehodna. V Pythonu to omogočata ukaza `exec` in `eval` (ter `execfile`). Na primer:

```
>>> ukazi = 'b = "ha"; c = (b+"-")*10+b'
>>> ukazi
'b = "ha"; c = (b+"-")*10+b'
>>> exec ukazi
>>> c
'ha-ha-ha-ha-ha-ha-ha-ha-ha-ha-ha'
>>> from math import *
>>> f = 'sin(x)+2*cos(3*x)'
>>> for i in range(10):
        x = i/10.; print(i, x, eval(f))

0 0.0 2.0
1 0.1 2.0105063949
2 0.2 1.84934056061
3 0.3 1.5387401432
4 0.4 1.11413385126
5 0.5 0.62089994194
6 0.6 0.110238284009
7 0.7 -0.365474521962
8 0.8 -0.757431340183
9 0.9 -1.02481737441
>>>
```


Program na datoteki

Če nameravamo program poganjati na različne načine, ga 'opremimo' tako, kot je storjeno na datoteki `pozdrav1.py`:

```
#!/usr/bin/python

ime = 'Janez'

def pozdrav():
    """Pozdrav()  izpise lep pozdrav
    osebi navedeni v spremenljivki  ime.

    V. Batagelj, januar 2009"""
    print('dober dan', ime)

if __name__ == '__main__':
    pozdrav()
else:
    print(pozdrav.__doc__)
```

. . . program na datoteki

```
>>> import pozdrav1
Pozdrav()  izpise lep pozdrav
   osebi navedeni v spremenljivki  ime.

   V. Batagelj, januar 2009
>>> pozdrav1.pozdrav()
dober dan Janez
```

in v ukaznem načinu

```
D:\Python\3.0>python pozdrav1.py
dober dan Janez

D:\Python\3.0>
```

. . . program na datoteki

```
>>> import sys
>>> sys.path
['D:\\Python\\2.3\\Lib\\idlelib', 'C:\\WINNT\\system32\\python23.zip',
'D:\\Python\\2.3', 'D:\\Python\\2.3\\DLLs', 'D:\\Python\\2.3\\lib',
'D:\\Python\\2.3\\lib\\plat-win', 'D:\\Python\\2.3\\lib\\lib-tk',
'D:\\Python\\2.3\\lib\\site-packages']
>>> sys.path.append('D:\\vlado\\work\\Python\\seminar')
>>> sys.path
['D:\\Python\\2.3\\Lib\\idlelib', 'C:\\WINNT\\system32\\python23.zip',
'D:\\Python\\2.3', 'D:\\Python\\2.3\\DLLs', 'D:\\Python\\2.3\\lib',
'D:\\Python\\2.3\\lib\\plat-win', 'D:\\Python\\2.3\\lib\\lib-tk',
'D:\\Python\\2.3\\lib\\site-packages', 'D:\\vlado\\work\\Python\\seminar']
>>> import pozdrav1
Pozdrav()  izpise lep pozdrav
   osebi navedeni v spremenljivki  ime.

   V. Batagelj, junij 2004
>>> pozdrav1.pozdrav()
dober dan Janez
>>>
```

Po popravkih

```
reload(pozdrav)
```

S posebnim programom, kakršen je npr. py2exe, lahko prevedeni program v Pythonu predelamo v izvršljiv program.

Porazdelitev besed v besedilu

```
import re
bes=open("c:\\Users\\Batagelj\\test\\python\\usher10.txt","rU")
b=bes.read(); bes.close()
i = b.find('\n',10+b.find('\n',b.find("*END*THE SMALL PRINT!")))
slovar = {}
for beseda in re.split('\W+',b[i:].lower()):
  if beseda in slovar: slovar[beseda] += 1
  else: slovar[beseda] = 1
for (n,kv) in enumerate(sorted(slovar.items(), \
  key=lambda x: (-x[1],x[0]))):
  (k,v) = kv
  print("%5i %20s %7i" % (n+1,k,v))
```

```
    1                  the     567
    2                   of     420
    3                  and     245
    4                    i     171
    5                    a     160
    6                   in     145
    7                   to     121
 ...........
 2043                  yes       1
 2044                youth       1
```

Vislice

```
# -*- coding: windows-1250 -*-
from random import seed, randint
def run(sezBesed):
   try:
      besede = open(sezBesed,'rU').readlines()
   except IOError: print("Težave z datoteko",sezBesed)
   else:
      seed(None)
      beseda = besede[randint(0,len(besede)-1)].strip().lower()
      vzorec = "?"*len(beseda)
      izbrane = ""; odkrita = False; krat = 5; narobe = 0; k = 0
      while narobe < krat:
         k += 1
         print("\n",k,". ugibaj = ",vzorec,sep='')
         print("   črke   =",izbrane,"\n")
         znak = input("črka = ")[0].lower()
         izbrane += znak; vzorecNov = ""
         for i,z in enumerate(beseda):
            if znak==z: vzorecNov += znak
            else: vzorecNov += vzorec[i]
         if vzorec==vzorecNov:
            narobe += 1
            print(narobe,". napačna črka",sep='')
         else: odkrita = beseda==vzorecNov
         if odkrita: break
         vzorec = vzorecNov
      print("\nBeseda =",beseda)
      print(["Obešen","Čestitke"][odkrita])
run(r'c:\Users\Batagelj\test\python\vislice\besede.txt')
```

Želvja grafika

Python uporablja za slikovni vmesnik sestav Tk, ki je dostopen s knjižnico tkinter. Ta vsebuje tudi podporo risanja. Ker je uporaba te knjižnice razmeroma zapletena, so za preprosta risanja dodali prijaznejšo knjižnico turtle. Leta 2006 je Gregor Lingl predstavil izpopolnjeno knjižnico xturtle, ki je osnova za izpopolnjeni `turtle` v Python 3.

Želvja grafika je ena izmed glavnih sestavin programskega jezika Logo, ki so ga razvili na MIT v drugi polovici šestdesetih let (Papert, Fuerzig). Logo se uporablja predvsem za zgodnje uvajanje otrok v programiranje. O želvji grafiki sta Abelson in diSessa napisala knjigo. Želvja grafika temelji na sledi, ki jo pri premikanju po ravnini za sabo pušča želvica.

Želvja grafika in Python 3

Pri interaktivnem delu z želvjo grafiko so težave. Rešitev je naslednja (povzeta po bytes):

V `c:\Python30` kliknemo z desno tipko na `pythonw` in izberemo možnost `Create Shortcut`. Sistem ustvari bližnjico `pythonw - Shortcut`. Preimenujemo jo (desni klik, `Rename`) v npr. `IDLE (turtle)`. Nato desno kliknemo na to datoteko in izberemo `Properties`. Polje `Target` spremenimo tako, da vsebuje

```
C:\Python30\pythonw.exe "C:\Python30\Lib\idlelib\idle.pyw" -n
```

Pri zagonu `IDLE (turtle)` se sedaj izpiše

```
  IDLE 3.0       ==== No Subprocess ====
```

Če želimo, da se bližnjica pojavi med drugimi možnosti za Python v seznamu programov, jo prestavimo (`move`) v področje

```
C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Python 3.0
```

Izbor ukazov – premiki in risanje

`forward(`d`)`	naprej za d
`fd(`d`)`	
`backward(`d`)`	nazaj za d
`bk(`d`)`	tudi `back(`d`)`
`right(`a`)`	desno za kot a
`rt(`a`)`	`"logo"`: 0 – S, 90 – V; `"standard"`: 0 – V, 90 – S
`left(`a`)`	levo za kot a
`lt(`a`)`	
`setposition(`xy`)`	premik na točko xy
`setpos(`xy`)`	tudi `goto(`xy`)`
`setx(`x`)`	sprememba koordinate x
`sety(`y`)`	sprememba koordinate y
`setheading(`a`)`	usmeri želvo v smeri (kot) a
`seth(`a`)`	
`home()`	v koordinatno izhodišče
`circle(`$r, \ldots$`)`	krog s polmerom r
`dot(`d, c`)`	pika premera d barve c
$i =$`stamp()`	odtis
`clearstamp(`i`)`	
`clearstamps(`n`)`	
`undo()`	prekliči zadnji želvji ukaz
`=speed(`s`)`	spremeni hitrost želvice 0 - 10, 1-počasi 10-najhitreje, 0-kar se da hitro

Izbor ukazov – stanje in koti

xy =`position()`	koordinate želvice
`pos()`	
a =`towards(`xy`)`	kot želvice proti točki
x =`xcor()`	koordinata x
y =`ycor()`	koordinata y
a =`heading()`	smer želvice
d =`distance(`xy`)`	oddaljenost želvice do točke
`degrees()`	koti se merijo v stopinjah
`radians()`	koti se merijo v radijanih
`showturtle()`	želvica postane vidna
`st()`	
`hideturtle()`	želvica se skrije
`ht()`	
p =`isvisible()`	ali je želvica vidna?
`shape(`N`)`	prikaz želve N: `"turtle"`, `"classic"`, `"arrow"`, `"circle"`, `"square"`, `"triangle"`, uporabniške
`resizemode(`m`)`	m: `"auto"`, `"user"`, `"noresize"`
`shapesize(`w, l, o`)`	raztega in debelina obrisa
`turtlesize()`	
`tilt(`a`)`	nagib – zasuk slike želve
`settiltangle(`a`)`	
a =`tiltangle()`	

Izbor ukazov – pero

`pendown()`	spusti pero
`pd()`	tudi `down()`
`penup()`	dvigni pero
`pu()`	tudi `up()`
`pensize(`d`)`	velikost peresa; tudi `width()`
t =`pen()`	vse lastnosti peresa
`isdown()`	ali je pero spuščeno
`color(`c_1, c_2`)`	$\equiv$ (`pencolor(`c_1`)`, `fillcolor(`c_2`)`)
=`pencolor(`c`)`	barva peresa
=`fillcolor(`c`)`	barva zapolnjevanja
`filling()`	zapolnjevanje ?
`begin_fill()`	začetek zapolnjevanja (barvanja notranjosti) lika
`end_fill()`	konec zapolnjevanja lika
`reset()`	počisti zaslon in začni nanovo
`clear()`	zbriši sledi
`write(`z`,...)`	napiše niz znakov z; `move=` `True`/`False`, `align=` `"left"/"center"/"right"`, `font=` (ime,velikost,oblika)

Izbor ukazov – dogodki

`onclick()`	
`onrelease()`	
`ondrag()`	
`begin_poly()`	začetek beleženja mnogokotnika
`end_poly()`	konec beleženja mnogokotnika
M =`get_poly()`	zadnji zabeleženi mnogokotnik
z =`Turtle()`	ustvari novo želvo z
z =`getturtle()`	vrne brezimno želvo
`getpen()`	
`clone(z)`	ustvari dvojnika želve z
`getscreen()`	
`setundobuffer()`	
`undobufferentries()`	
`delay()`	
`tracer()`	
`update()`	
`listen()`	
`onkey()`	
`onscreenclick()`	
`onclick()`	
`ontimer()`	

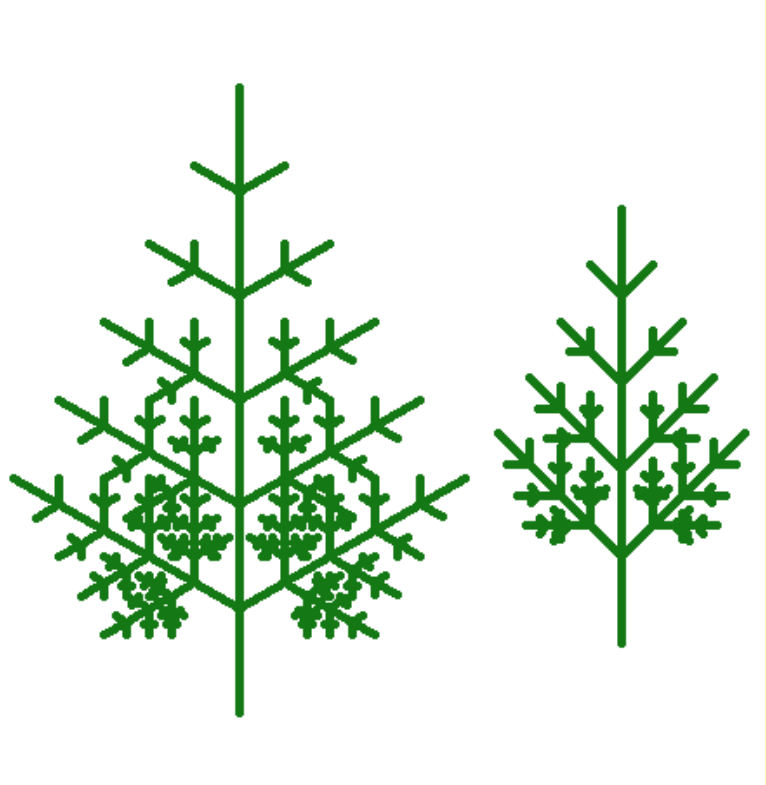
Izbor ukazov – okno

`bgcolor(`c`)`	barva ozadja
`bgpic(`P`)`	slika v ozadju
`clearscreen()`	pobriši zaslon
`clear()`	
`resetscreen()`	začni nanovo
`reset()`	
`screensize(`w, h, c`)`	želvje okno naj bo široko w, visoko h in barve c
`setworldcoordinates(`P`)`	svet: $P = (llx, lly, urx, ury)$
`mode(`m`)`	m: `"logo"`, `"standard"`
`colormode(`m`)`	način zapisa barv RGB: 1.0 ali 255
`getcanvas()`	
s =`getshapes()`	seznam želvjih oblik
`register_shape(`N, M`)`	dodaj novo obliko želve M z imenom N
`addshape()`	ali sličico v zapisu GIF
s =`turtles()`	seznam želv
d =`window_height()`	višina okna
d =`window_width()`	širina okna
`bye()`	končaj z risanjem, zapri okno
`exitonclick()`	ob kliku na okno ga zapri
`setup()`	
`title(`N`)`	N postavi za naslov okna

Primeri

```
>>> from turtle import *
>>> reset()
>>> shape()
'classic'
>>> shape("turtle")
>>> fd(100); rt(90)
>>> ena = getturtle(); ena.fd(50)
>>> dva = Turtle(); dva.rt(120)
>>> dva.pencolor("blue"); dva.shape("circle")
>>> dva.fd(100)
>>> bye()
```


Primer – Smreka

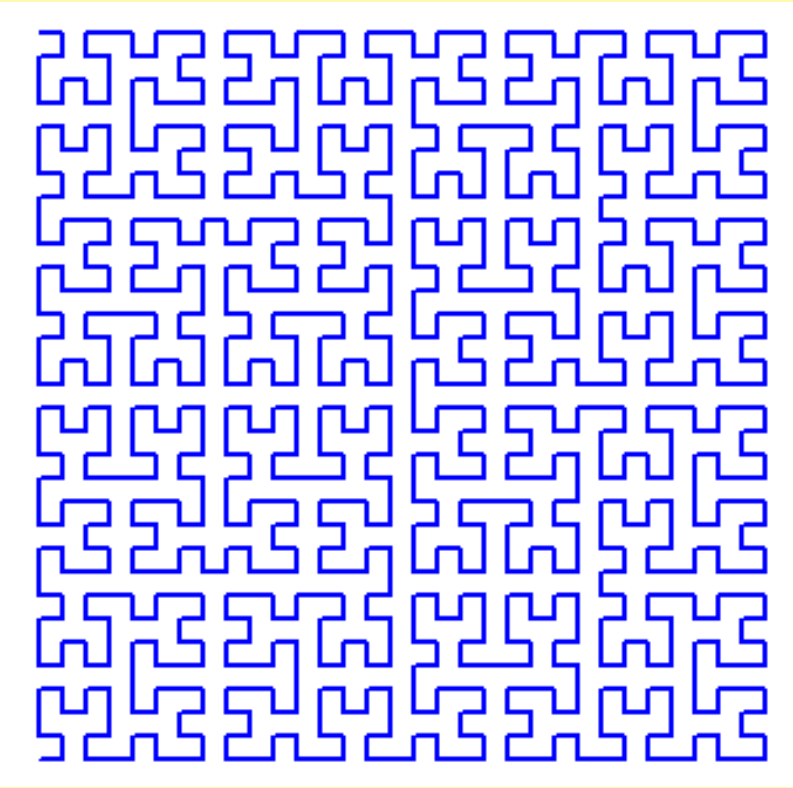


datoteka `smreka.py`

```
from turtle import *

def smreka(d,a,n):
  if n > 0:
    fd(d); rt(a); smreka(d/2,a,n-1)
    lt(a); smreka(d,a,n-1)
    lt(a); smreka(d/2,a,n-1)
    rt(a); pu(); bk(d); pd()

def NovoLeto():
  reset(); speed(0); ht(); seth(90)
  colormode(255);
  pencolor((20,120,20)); pensize(5)
  pu(); setpos(-80,-190)
  pd(); smreka(60,60,6)
  pu(); setpos(140,-150)
  pd(); smreka(50,45,5)
  exitonclick()

NovoLeto()
```

```
>>> import sys
>>> wdir = r'c:\users\Batagelj\test\Python\turtle'
>>> sys.path.append(wdir)
>>> import smreka
```

Primer – Hilbertova krivulja

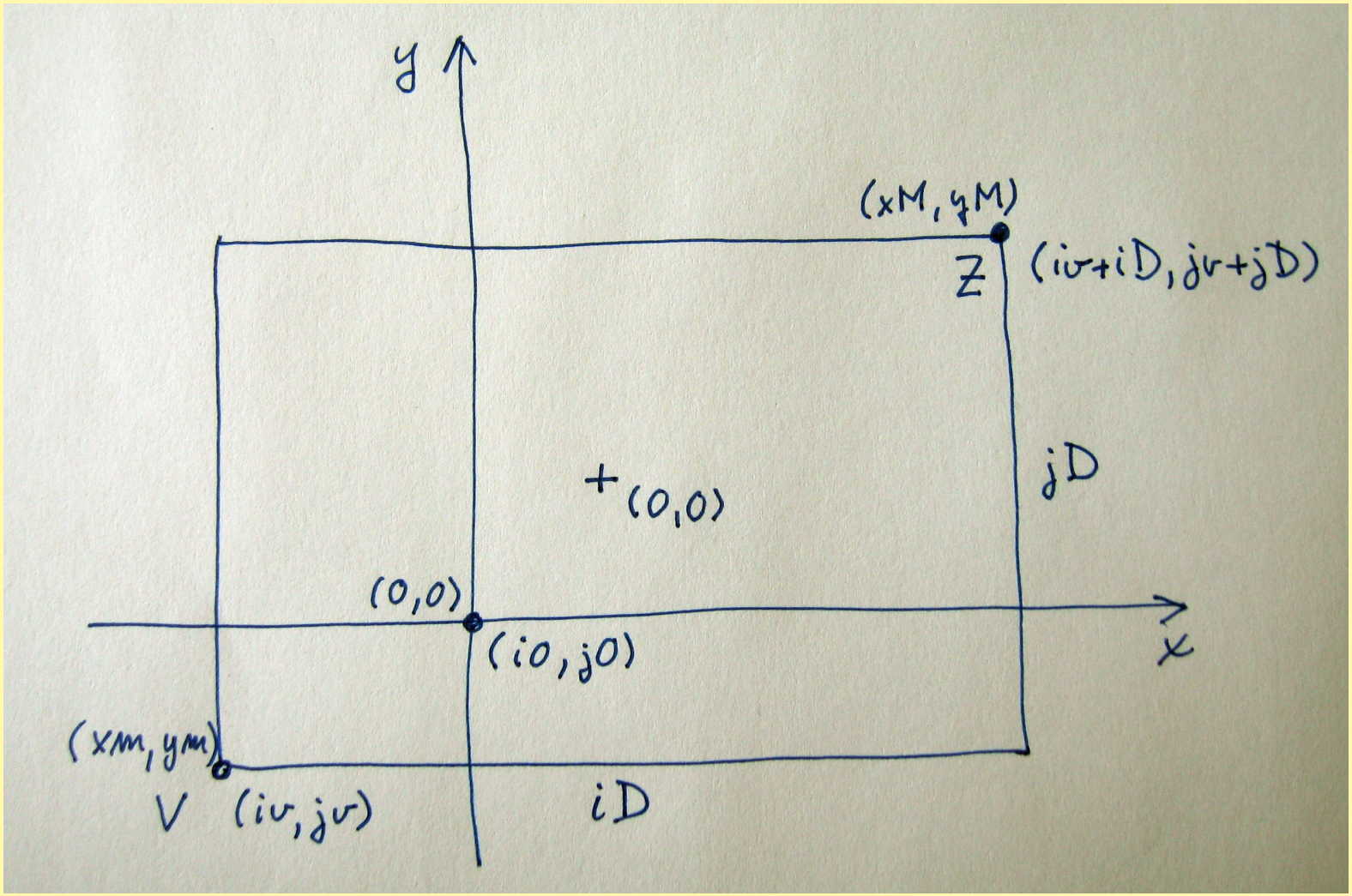


datoteka `Hilbert.py`

```
from turtle import *

def Hilbert(n,a,h):
  if n==0: return
  rt(a); Hilbert(n-1,-a,h); fd(h)
  lt(a); Hilbert(n-1,a,h); fd(h)
  Hilbert(n-1,a,h); lt(a); fd(h)
  Hilbert(n-1,-a,h); rt(a)

def RisHi():
  reset(); speed(0); ht()
  pu(); setpos(-160,-160); pd()
  seth(90); pensize(2)
  colormode(255); pencolor("blue")
  Hilbert(5,90,10)
  exitonclick()

RisHi()
```

```
>>> import sys
>>> wdir = r'c:\users\Batagelj\test\Python\turtle'
>>> sys.path.append(wdir)
>>> import Hilbert
```

Primer – Risanje funkcije



Primer – Risanje funkcije

Koordinate (x, y) ravnine $\mathbb{R}^2$ in zaslonske koordinate (i, j) so med seboj linearno povezane: $i = ax + b$ in $j = Ay + B$.

Določimo a in b; za A in B gre enako. Če vstavimo v $i = ax + b$ točki V in Z, dobimo $iv = a \cdot xm + b$ in $iv + iD = a \cdot xM + b$.

Rešitvi a in b tega sistema enačb sta

$$a = \frac{iD}{xM - xm} \quad \text{in} \quad b = iv - a \cdot xm = iv - \frac{iD \cdot xm}{xM - xm}$$

oziroma končno, če označimo a z ei (enota na i)

$$i = ei \cdot (x - xm) + iv \quad \text{in} \quad x = \frac{1}{ei}(i - iv) + xm$$

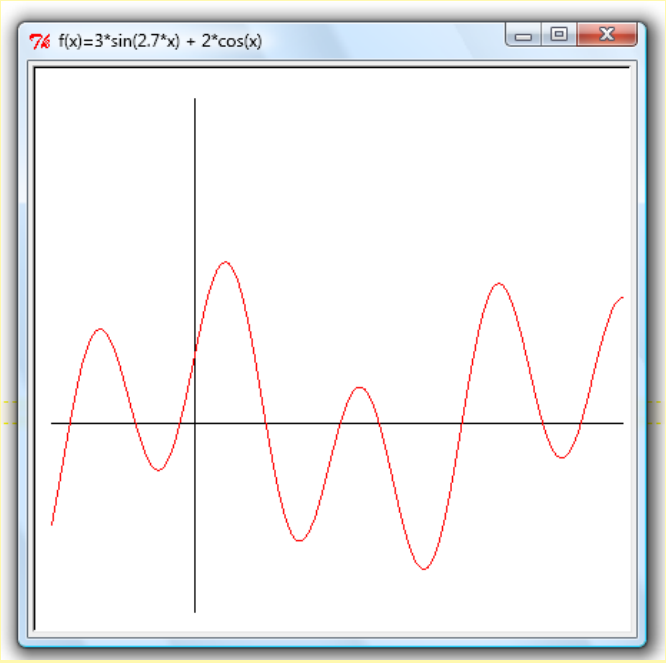
Podobno velja:

$$ej = \frac{jD}{yM - ym}, \quad j = ej \cdot (y - ym) + jv \quad \text{in} \quad y = \frac{1}{ej}(j - jv) + ym$$

Za zaslonski koordinati $(i0, j0)$ koordinatnega izhodišča dobimo

$$i0 = iv - ei \cdot xm \quad \text{in} \quad j0 = jv - ej \cdot ym$$

Primer – risanje funkcij



datoteka `funkcija.py`

```
from turtle import *
from math import *

mode("logo")
f = "3*sin(2.7*x) + 2*cos(x)"
iv = -200; jv = -180; iD = 400; jD = 360
xm = -2.5; xM = 7.5; ym = -5.5; yM = 9.5
ei = iD/(xM-xm); ej = jD/(yM-ym)
i0 = iv-ei*xm; j0 = jv-ej*ym

reset(); title("f(x)="+f)
ht(); pencolor("black")
pu(); setpos(iv,j0); seth(90); pd(); fd(iD)
pu(); setpos(i0,jv); seth( 0); pd(); fd(jD)
x = xm; j = round((eval(f)-ym)*ej)+jv
pu(); setpos(iv,j); pd(); pencolor("red")
for i in range(1,iD+1):
  x = i/ei + xm
  j = round((eval(f)-ym)*ej)+jv
  setpos(i+iv,j)
exitonclick()
```

```
>>> import sys
>>> wdir = r'c:\users\Batagelj\test\Python\turtle'
>>> sys.path.append(wdir)
>>> import funkcija
```