Python - motivacija

- Pet osnovnih zahtjeva na sustav za znanstveno računanje:
  - Potpuni programski jezik.
  - Interaktivan.
  - Integrirana grafika.
  - Široka baza biblioteka.
  - Široka baza korisnika.

- U obzir dolaze Python, R, Matlab (i izvedenice).

Zašto Python

- Scipy-lectures.org, 1.1.1.2 i 1.1.1.3 (http://www.scipy-lectures.org).
- Isto je ScipyLectures-simple.pdf (SLS), str. 5-6, (https://www.scipy-lectures.org/_downloads/ScipyLectures-simple.pdf).

- Tri cjeline (SLS, str. 6-8):
  - Python (jezik) u užem smislu ("jezgra") s osnovnim modulima.
  - Ipython.
  - Ključne numeričke i grafičke biblioteke (Numpy, Scipy i MatPlotLib) (+ dodatni paketi).

Python - jezik

- Povijest:
  - Quido van Rossum, nizozemac, ~1980.
  - prva verzija, 1989.
  - Python 2.0, 2000.
  - Python 2.6 i Python 3.0, 2006.
  - Python 2.7 i Python 3.1, 2009.
  - Dalje samo 3.x, dok je 2.7 podržan do 2020.
  - Python 3.x nije kompatibilan unatrag s 2.x iako na prvi pogled nema razlike. Npr. u Pythonu 2.x je 5/2=2, a u Pythonu 3 je 5/2=2.5, dok je uvijek 5//2=2.

Python - jezik

- Proceduralna i objektna paradigma (OO – objektno orjentirano):
  - https://en.wikipedia.org/wiki/Procedural_programming.
  - https://www.codenewbie.org/blogs/object-oriented-programming-vs-functional-programming.
- U svim računalnim programima postoje dvije komponente:
  - podaci (ono što program zna da ima).
  - ponašanja (ono što program može napraviti na podacima).
- OO paradigma: Čvrsto povezuje podatke i pripadna ponašanja na jednom mjestu (u tzv. **objektu**).
- Proceduralna paradigma: Podaci i ponašanja (postupci, algoritmi) su strogo odvojeni.

Python - jezik

- Python podržava i jedan i drugi način programiranja - jako zgodno za učenje i za eventualni prijelaz na OO.
- Ipak, Python je izrazito **objektno orjentiran**, ali ne kroz OO programiranje već kroz dosljedno proveden objektni ustroj!
- U Pythonu **sve je objekt** i to ima dalekosežne posljedice.

Što je objekt (u Pythonu)?

- Objekt je Pythonova **apstrakcija** za odredjeni skup podataka (u najširem smislu). Objekt ima:
  - **Identitet** (fizički postoji u memoriji i ima svoju adresu; naredba id()).
  - **Tip** (pojam koji odredjuje vrstu objekta; popis svih svojstava (atributa) zadaje sve objekte istog tipa).
  - **Atribute** (svojstva i metode, s tim da Python ne pravi jaku razliku izmedju atributa i metoda).
  - Vrijednost (sadržaj, podaci u užem smislu).
- Objekti mogu imati neka 'opća' svojstva: **izmjenjiv** (mutable) / **nezmjenjiv** (unmutable), '**pozivljiv**' (callable) / '**nepozivljiv**' (non-callable), **pobrojiv** (iterable), **slijedni** (sequence), …
- Tip se dobiva naredbom type(<ime_objekta>), npr.:
  type('abcd123') → str.
- Identitet se dobiva naredbom id(<ime_objekta>) npr.:
  id('abcd123'). → 140399258856928.

Imenovanje objekata (*assignment*)

- **Imenovanje** objekata se vrši naredbom **dodjeljivanja** imena (**asignment**)
- Npr.:

  a = 7 --> Lijevo je IME, desno je OBJEKT; povezuje (bind) ime s objektom.
- Općenito:

  <ime> = <izraz>
- Npr.:a

  b = 42 → Ime **b** se dodjeljuje objektu **42**.
  c = 42 → I ime **c** se dodjeljuje objektu **42** (a is b → True).
  b+5 → Ispisuje 47, ali se veza izmedju b i 42 ne mijenja.
- b = complex(3,-2) → Veza imena b s objektom 42 se raskida, ali ime c ostaje
  b+5 → Ispisuje (8-2j)

- Python je **dinamički tipizirani** jezik, tj. prije upotrebe **imena ne treba deklarirati**, tj. pridjeliti im tip. Posljedično, iz imena se ništa (unaprijed) ne može zaključiti o tipu!

Varijante pridjeljivanja

- Višestruko pridjeljivanje (raspakiravanje):
  - x, y, z = 2, 4, 7
  - S desne strane može biti **bilo koji pobrojivi objekt**. S lijeve strane su imena odvojena zarezom. Jedna, a ne više naredbi.

- Prošireno pridjeljivanje:
  - x = x+2 → Na desnoj strani stvara novi objekt i pridružuje mu ime x.
  - x+=2 → Nastoji ažurirati postojeći objekt; ponašanje ovisi o tome da li je x izmjenjiv ili nije!!
- Npr. x = [1, 2, 3] → Lista, izmjenjivi objekt.
  - y = x → Novo ime za jednu te istu listu.
  - x+=[5, 7] → x = [1, 2, 3, 5, 7] = y !!
  - id(x)==id(y) → True.
- Npr. x = (1, 2, 3) → n-torka, neizmjenjivi objekt.
  - y = x → Novo ime za jednu te istu n-torku.
  - x+=(5, 7) → x = (1, 2, 3, 5, 7), y = (1, 2, 3), tj. imena x i y sada pokazuju na različite objekte!

Varijante pridjeljivanja, nast.

- Takodjer:
  - X = (1, 3, 5) → n-torka, nepromjenjiv.
  - X[1] = 7 → Javlja pogrešku: TypeError: 'tuple' object does not support item assignment.
  - X = X[0:1]+(7,)+X[2:3] → X = (1, 7, 5), proizvodi novi objekt, a veza sa starim se raskida.

Jednostavni tipovi

- **Brojevi,** tipovi **int, float, complex** (svaki ima svoje računske operacije; pazi na cjelobrojno dijeljenje u Py2 i Py3).
- **Logičke vrijednosti,** tip **bool** (True, False):
  - bool(broj) = True ako je broj != 0, = False ako je broj = 0.
  - bool(niz) = True ako niz nije prazan, = False ako je niz = ''.
- **nizovi znakova** (tip **str):**
  - x = 'abc123' ili x = “abc123”.
  - Slijedni tip (dopušta pristup preko indeksa, x[2] → 'c'.
  - Neizmjenjiv.
  - Mnogobrojni atributi/metode.

Kolekcije objekata (složeni tipovi)

(engl. containers, collections)

- Grupiranje više objekata koji dijele zajedničku poveznicu ili smisao.
- Ugradjene kolekcije opće namjene:
  - **lista**, **n-torka** } **Slijedne** kolekcije (uz nizove znakova). Pristup elementima preko rednog broja (indeksa).
  - **rječnik**, **skup** } **Asocijativne** kolekcije. Pristup elementima putem ključa.
- Sve standardne kolekcije su pobrojive (tj. imaju **standardni protokol obilaska** (SPO) elemenata) → univerzalnost (zapravo podržavaju objekt tipa *iterator* koji ima metodu *next*).
- **Osnovne operacije** nad kolekcijama:
  - **ispitivanje pripadnosti** (operator **in**).
  - **obilazak elemenata** putem SPO-a, koristi se u for petlji.
  - NAP. Ako objekt podržava SPO onda podržava i operator **in**!!

Kolekcije objekata (složeni tipovi), nast.

(engl. containers, collections)

- **Dodatne operacije.** Ugradjene kolekcije:
  - imaju operator jednakosti (**==**)
  - mogu se interpretirati kao logičke vrijednosti
  - imaju broj elemenata (ugrađena funkcija **len**)

Slijedne kolekcije, nizovi

(engl. sequences)

- **N-torka:**
  - Okuplja objekte različitih tipova, npr. ('pero', 1, True, 7.2,).
  - Stvaranje n-torki:
    - konstruktor **tuple**, ime = tuple(iterable).
    - izraz za tvorbu (operator **tvorbe** je zarez "**,**", koji na kraju može ostati, a kod jednočlane n-torke mora ostati), npr. X = (3, 5, 1,).
  - Neizmjenjiva.
  - Podržava operatore "+" i "+=" .
  - Koristi se za zadavanje više vrijednosti odjednom.
  - Operacije: **nadovezivanje** i **ponavljanje**, npr.:
    a = 2*('a', 'b') + (1, 7, 3) → a = ('a', 'b', 'a', 'b', 1, 7, 3).
  - metode: **index**(), **count**(), npr.:
    a.index('b') daje 1, dok a.count('b') daje 2.

Slijedne kolekcije, nizovi, nast.

(engl. sequences)

- **Lista**
  - Slična n-torki, ali izmjenjiva (mutable).
  - Zadaje se uglatim zagradama (zarez na kraju može i ne mora biti).
  - Stvaranje lista:
    - Konstruktor **list**, ime = list(iterable).
    - Izraz za **tvorbu**, npr. ime = [4, 'a', True, 1.2, 'rtg'].
    - **Obuhvaćajućim izrazom** iz drugog objekta:

      lista1 = [x for x in <pobrojivi_objekt> if <uvjet>], npr.:

      b = [x for x in ime if type(x) is str] → b = ['a', 'rtg'].
  - Operator += proširuje postojeći objekt.
  - **Operacije** nad listama:
    - Izdvajanje elemenata **indeksiranjem** i **izrezivanjem**.
    - Transformiranje **nadovezivanjem** i **ponavljanjem.**
  - **Metode** lista: append, index, remove, count, insert, reverse, extend, pop, sort.

Slijedne kolekcije, nizovi, nast.

(engl. sequences)

- Indeksni operator “[ ]” (digresija):
  - x[0] prvi elt, x[5] šesti elt.
  - x[-1] zadnji elt, x[-2] predzadnji elt ...
  - x[6:11] elt-i od 7 do 11 (**zadnji ima indeks 10!!**).
  - x[6:11:2] kao i gore, ali s korakom 2.
  - x[:5:2] od prvog do petog s korakom 2.
  - x[5::2] od šestog do zadnjeg s korakom 2.
  - x[:] - cijeli niz.
  - x[::-1] cijeli niz, unatrag.

- Nap.: Tipična upotreba:
  - N-torke: različiti, ali povezani podaci.
  - Liste: istovrsni, nezavisni podaci.

Slijedne kolekcije, rasponi

- **Raspon**, npr. a = xrange(3,10000):
  - Virtualne kolekcije.
  - Neizmjenjivi.
  - Elt-e stvaraju na zahtjev pa ne troše memoriju.
  - NAP: U Py2 naredba *range* proizvodi listu. U Py3 postoji samo *range* koji radi kao xrange.

Asocijativne kolekcije

- **rječnik (dictionary):**
  - Kolekcija čijim elementima se pristupa putem **neizmjenjivih ključeva**
  - Stvaranje rječnika:
    - Izraz za tvorbu su **{}**, npr.: a = {'ana': 45, 'ante': 73, 'pero': 87}
    - Konstruktor **dict,** npr.: a1 = dict([('pero', 87), ('ante', 73), ('ana', 45)])
  - Pristup elementima je putem indeksnog operatora [], ali umjesto rednog broja dolazi ključ, npr.: a1['ante'] → 73
  - Nema indeksa!!
  - Redoslijed obilaska postoji (pobrojiva kolekcija), ali nije predvidiv.
  - Približno konstantno vrijeme pristupa, umetanja i brisanja.
  - Mnogobrojne metode: clear, items, pop, viewitems, copy, iteritems, popitem, viewkeys, fromkeys, iterkeys, setdefault, viewvalues, get, itervalues, update, has_key, keys, values

Kontrola tijeka programa

- if-elif-else
  - if <logicki uvjet>: # zaglavlje
    <naredbe tijela> # tijelo

  - if <uvjet1>:
    <tijelo1>
    elif <uvjet2>:
    <tijelo2>
    …
    else:
    <tijeloAlt>

Kontrola tijeka programa, nast.

- Uvjetni izraz:
  - Rezultat **uvjetnog izraza** je jedna od dvije ponudjene vrijednosti ovisno o rezultatu **logičkog izraza.** Konkretno:
  - <rezultat-ako-istina> if <logički-uvjet> else <rezultat_ako_laž>.
  - Radi isto što i

```
if A:
        x=A1
else
        x=B1
```

  - OK ako tijela u strukturi if-else sadrže samo pridruživanje jednog te istog imena prema jednoj od dvije moguće vrijednosti.
  - x=A1 if A else B1

Kontrola tijeka programa, nast.

- Petlja **while** (nepoznat broj ponavljanja):
  - while <uvjetni izraz>
    <tijelo>

- Petlja **for** (poznat broj ponavljanja):
  - for <upravljacko_ime> in <**pobrojivi_objekt**>
    <tijelo>

- Npr.:

```
popis = ['pero', 'ante', 1234]
for i in popis:
    print i
```

daje:

```
pero
ante
1234
```

Kontrola tijeka programa, nast.

- Konstrukt **enumerate**: iterator za indeks i vrijednost pobrojivog (iterabilnog) objekta (radi se o tuple-ima i raspakiravanju, vidi https://www.afternerd.com/blog/python-enumerate/).
  - for n, x in enumerate(**pobrojivi_objekt**)
    <tijelo>

- Npr:

```
popis = ['pero', 'ante', 1234]
for n, ime in enumerate(popis):
    print ime, 'ima redni broj', n
```

daje:

```
pero ima redni broj 0
ante ima redni broj 1
1234 ima redni broj 2
```

Funkcije

- Funkcija je programska cjelina (skup naredbi) koji obavlja neku dobro definiranu zadaću.
- Funkcija prima ulazne podatke putem (**ulaznih**) **argumenata**
- 2 načina da funkcija obavi svoju zadaću:
  - izračunavanjem povratne vrijednosti, ili
  - kroz popratne učinke (engl. *side efects*, npr. crtanje, ispis).
- Funkcije mogu biti '**samostojeće**' i **članske funkcije** (**metode**) nekog tipa. Mi ćemo pisati samo prve, a koristiti (obilno) obje.
- **Definicija** funkcije (stvara funkcijski objekt i daje mu ime):
  - def imeFunkcije(<argumenti>)
    <kod_funkcije>
- **Poziv** funkcije:
  - imeFunkcije() ili
  - imeFunkcije(argumenti).

Funkcije, nast.

- Npr. za funkciju:

```
def sredVr(x)
    print(sum(x)/len(x))
```

- poziv

```
temp = [7., 3., 11., 1.]
sredVr(temp)
```

ispisuje 5.5.

- x je **formalni** argument, a temp je **stvarni** argument.

Argumenti funkcija i povratne vrijednosti

- Služe za prijenos podataka **u** funkciju.
- U pozivu treba paziti na broj, redoslijed i tip argumenata (iako to Ptyhon neće kontrolirati unaprijed).
- Ako se želi objektu promijeniti vrijednost putem formalnog argumenta, jedini način je pozvati **njegovu** metodu. No, pri tom stvarni argument mora biti izmjenjivog tipa!!
- Python argumente prenosi **dodjeljivanjem**.

- **Povratne** vrijednosti f-je: Naredba

  return <ime ili izraz>

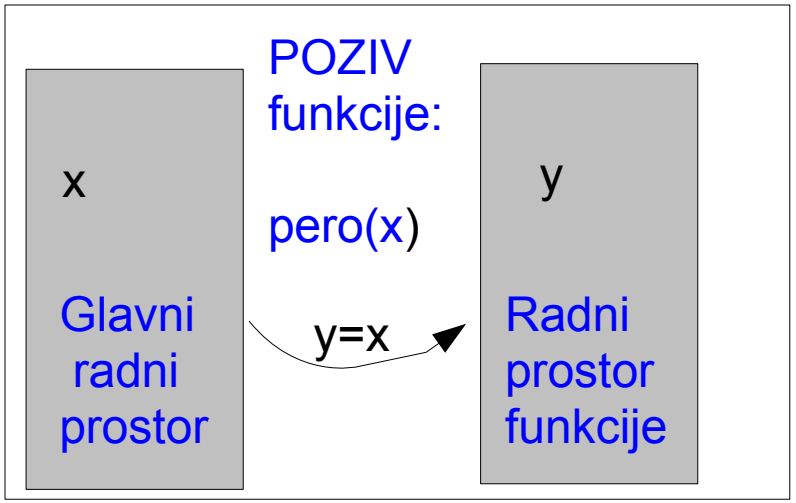
  trenutno pekida izvodjenje funkcije i daje **povratni objekt**, kojemu se u pozivnom kodu **pridjeljuje ime**. Ako iza return nema ničega, vraća se posebna vrijednost None.

Prenošenje argumenata u Matlabu i Fortranu

(digresija)

- Matlab

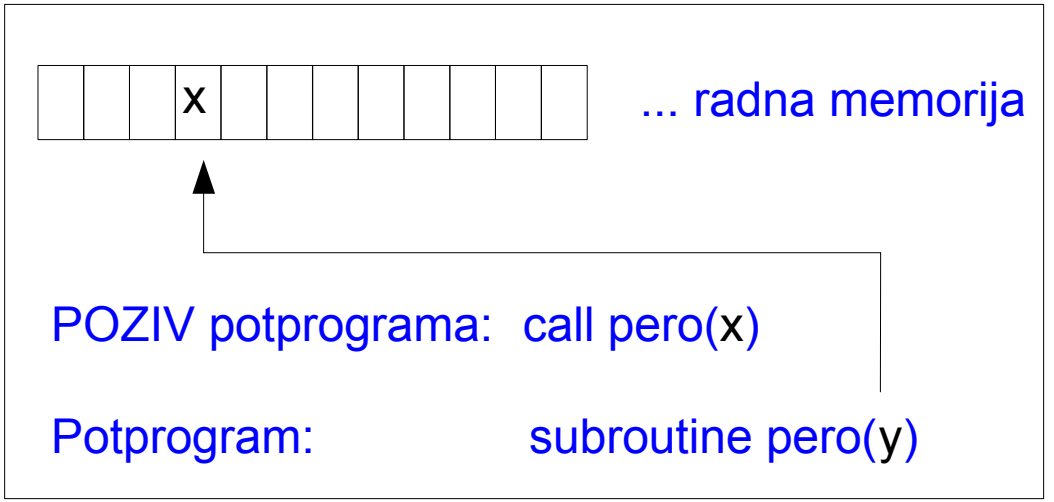
```
function  pero(y)
  y = 2
  return
```



- x se kopira
- funkcija radi s **kopijom**
- izvorni x ostaje netaknut
- prenose se vrijednosti (by value)

- Fortran

```
subroutine  pero(y)
  y = 2
  return
  end
```



- y je fiktivna varijabla (zapravo pokazivač)
- u trenutku poziva y pokazuje na x
- funkcija radi **izravno** s x (iako ga zove y)
- u gornjem primjeru x poprima vrijednost 2
- prenose se adrese (by reference)

Imenovani argumenti i argumenti s podrazumijevanim vrijednostima

- Npr. funkcija definirana s:

```
def fja(a1, a2, a3, a4)
```

- se može pozvati s:

```
fja(x1, x2, x3, x4)                  # Poštuje se broj i redoslijed.
fja(x1, a3=x3, a4=x4, a2=x2)         # Nakon imenovanog, samo imenovani!
```

- ali ne i s:

```
fja(x1, a2=x2, x3, x4)               # Pogrešno.
```

- Može se definirati i ovako:

```
def fja(a1, a2, a3=xp3, a4=xp4)
```

- a pozivati s:

```
fja(x1, x2)                          # Podrazumijeva se a3=xp3, a4=xp4.
fja(x1, a3=x3, a4=x4, a2=x2)         # Kao i gore.
```

- ali ne i s:

```
fja(x1, a2=x2, x3)                   # Pogrešno.
```

Imenovani argumenti i argumenti s podrazumijevanim vrijendostima, nast.

- Imenovane argumente treba koristiti suzdržano jer povećavaju ovisnost izmedju funkcije i pozivnog koda.
- Podrazumijevane vrijednosti se vezuju (dodjeljivanjem konkretnih objekata) prilikom definiranja funkcije!!! Dakle, radi se o **jednom te istom** objektu. Zato treba koristiti neizmjenjive objekte ili None, pa u tijelu funkcije napraviti provjeru.
- Npr.:
```
def dodajListi(x,lista=[])
    lista.append(x)
    return list
```
- Pozivi:
```
dodajListi(1)      → [1]      # očekivano
dodajListi('a')    → [1, 'a']  # neočekivano!
```

Članske funkcije tipova (metode)

- Većina rečenog do sada vrijedi i dalje.
- Metode se mogu pozvati na **dva načina**. Npr., funkcija *index* tipa *str* vraća indeks podniza:

```
x = 'abcd'
x.index('c') → 2          # poziv u duhu OO paradigme (preferirani!)
str.index(x,'c') → 2      # poziv u duhu proceduralne paradigme.
```

- Može i izravno:

```
'abcd'.index('c') → 2       # poziv u duhu OO paradigme (preferirani!)
str.index('abcd','c') → 2   # poziv u duhu proceduralne paradigme.
```

- U prvom slučaju se podrazumijeva da je 'abcd' prvi argument u pozivu, tj. funkcija po defaultu radi na 'svom' objektu. Obje funkcije rade isto i imaju isti opis, ali nisu iste, čak ni istog tipa:

```
- x.index?
  S.index(sub [,start [,end]]) -> int
  Type:      builtin_function_or_method.
- str.index?
  S.index(sub [,start [,end]]) -> int
  Type:      method_descriptor.
```

Moduli i paketi

- Dva oblika programskih cjelina:
  - **Modul** je (pojedinačna) datoteka programskog koda s definicijama srodnih konstanti, funkcija (i/ili razreda).
  - **Paket** je skup modula u istom imeniku koji može sadržavati I podimenike s podpaketima (subpackages).
- Module i pakete uključujemo naredbom **import**, npr. naredba

  import sys

  stvara pristupni objekt (sys) u trenutnom prostoru imena. Atributi tog objekta daju pristup sadržaju modula, bez zadiranja u trenutni prostor imena (analogija s datotečnim sustavom).
- razne varijante naredbe **import:**

```
import matplotlib as mpl
import numpy.random as rnd
from numpy import random as rnd
from numpy.random import gamma
from numpy import *              # Ovaj oblik se obeshrabruje !!
```

- Python učitava modul samo prvi put!! Za ponovo učitati modul treba upotrijebiti funkciju **reload**(modul), npr. reload(sys).

Lokalna i globalna imena, prostori imena

(namespaces, digresija)

- Globalna: imena u modulu izvan tijela funkcije --> **globalni prostor imena.**
- Lokalna: imena kojima se vrijednost dodjeljuje unutar tijela funkcije (uključuju i formalne ragumente) --> **lokalni prostor imena.**
- Globalna imena **su vidljiva** u funkcijama, no mogu biti '**prekrivena**' lokalnim imenima, npr.:

```
a = 1; b =’pero’          # globalna imena
def fja(x)
    z = a+3               # Koristi se globalno ime a
    b = 7+x               # Lokalno ime b
    return b              # Nema utjecaja na globalno ime b!!!
```

- NAP.: postoji i **ugradjeni prostor** imena (**built-in** name space, import __builtin__), koji sadrži, npr., int, abs, len, ...,

IPython

- Ipython je napredno **interaktivno sučelje** za Python (s naredbenom linijom, tj. *command-line interface*), napredni terminal, konzola (slično kao u Matlabu). *Command line REPL (read-eval-print loop) environment*.
- Osnovne značajke:
  - Nakon pokretanja dobije se **kratki podsjetnik**:

    ? -> Introduction and overview of IPython's features.
    %quickref -> Quick reference.
    help -> Python's own help system.
    object? -> Details about 'object', use 'object??' for extra details.
  - **Ulazi** (interaktivne naredbe) se pamte u **listi In**, a **izlazi** (rezultati) u **rječniku Out.**
  - **Nadopuna sintakse** pomću tipke TAB radi u svim 'razumnim' situacijama. Pogodno za istražiti strukturu objekta, vidjeti atribute, ...
  - Pristup **dokumentaciji** putem znaka '**?**', npr.:

    ?objekt ili objekt?
    ?objekt.atribut ili objekt.atribut?
    ?funkcija ili funkcija? (i funkcija je objekt)

    daje osnovne informacije. Kada je moguće, **??** daje proširene informacije.

Ipython, nast.

- **Magične funkcije** (*magic functions*); rade unutar IPythona, slično kao naredbe shell-a. Najčešće:
  - **run** program.py Izvodi program.py unutar IPythona. Objekti koje program proizvede ostaju u interaktivnom prostoru imena!
  - **who** Ispisuje imena iz interaktivnog prostora.
  - **whos** Detaljniji ispis, vrlo korisno!! (Najčešće nas zanima tip nekog objekta ili vrijednost neke varijable.)
  - **whos** tip Ispisuje samo objekte zadanog tipa.
  - **reset** Briše sve iz interaktivnog prostora imena.
  - **reset** -f Isto, ali bez pitanja (*forced*).
  - **reset -f** array Kao i gore, ali samo NumPy arrays.
  - **reset_selective -f** ime1 ime2 Briše zadana imena.
  - **timeit** Mjeri vrijeme potrebno za izvršavanje.
- Naredbe **shella** su dostupne ali ih treba započeti znakom '!'. Npr. !cp pero.txt ante.txt.
- Neke naredbe, npr. ls, pwd, rade i bez uskličnika.

Znanstveno računanje u Pythonu (Scientific Computing)

- Ključne biblioteke (moduli):
  - **NumPy** definira podatkovne strukture za učinkovit rad s poljima.
  - **SciPy** kolekcija biblioteka s funkcijama za znanstveno računanje u mnogim područjima; temelji se na NumPy.
  - **MatPlotLib** biblioteka funkcija za vizualizaciju (daje 2D i 3D slike visoke kvalitete, tzv. **publication-quality**).

NumPy

- import numpy as np
- **NumPy polje** je objekt tipa **ndarray**:
  - **homogeno**, tj. sadrži istovrsne elemente.
  - **fiksne duljine** (broj elt-a se ne može mijenjati).
  - osnovni **atributi**: shape, size, ndim, nbytes, dtype.
- Npr.:

```
x = np.array([[1, 2, 3,],[4., 5., 6]])   →  array([[1., 2., 3.],
                                                   [4., 5., 6.]])
type(x)                                   →  numpy.ndarray
x.shape                                   →  (2, 3)
x.size                                    →  6
x.ndim                                    →  2
x.dtype                                   →  dtype('float64')
whos                                      →
Variable   Type      Data/Info
------------------------------
 x         ndarray   2x3: 6 elems, type `float64`, 48 bytes
```

IEEE aritmetika pokretnog zareza

(floating point, FP, digresija)

- opće prihvaćeni standard za aritmetiku na računalu
- **ideja**: ako su x, y ∈ FP, te ¤ ∈ {+,-,*,:,√}, tada je rezultat x¤y FP broj koji bi se dobio **egzaktnim računom** i naknadnim **zaokruživanjem** u FP broj. (Dakle **najbolje moguće**.)
- **dodatni zahtjev**: Za x, y ∈ FP, rezultat x¤y je **uvijek** ∈ FP, eventualno posebne vrste (Inf, -Inf, NaN)

- IEEE aritmetika je ugradjena unutar tipa np.float!!
- Funkcija np.finfo daje informacije o FP tipovima:
  - np.finfo(np.float64).max → 1.7976931348623157e+308 (realmax)
  - np.finfo(np.float64).tiny → 2.2250738585072014e-308 (realmin)
  - np.finfo(np.float64).eps → 2.220446049250313e-16 (**strojni epsilon**, relativna udaljenost između dva **susjedna** broja u FP sustavu

IEEE aritmetika pokretnog zareza

(floating point, FP, digresija)

- Imena **inf** i **nan** ne postoje unutar samog Pythona, već kao np.inf i np.nan, a mogu se zadati i kao inf = float('inf') te nan = float('nan').
- Inf = FP broj koji nastaje ako egzaktni rezultat premašuje realmax.
- NaN = FP broj koji nastaje iz operacija koje nemaju određeni rezltat, npr. 0/0, ∞ - ∞. **N**ot **a** **N**umber, jako koristan objekt!!
- Npr.:
  - 1/np.inf → 0, np.array(1.)/0 → inf, 1/np.nan → nan
  - (Skoro) svaka operacija koja uključuje NaN-ove rezultira s NaN!! **Vrlo korisno**!!!
  - Iznimke: 1**np.nan = 1, np.nan**0 = 1

NumPy, nast.

- Redoslijed elemenata u memoriji: default je 'raw' (kao u C-u), ali se može zadati da bude i 'column'.
- Kreiranje polja:
  - np.array(pobrojivi objekt), npr. :
    x = np.array([1, 2, 3]); → x.ndim=1, x.shape=(3,)
    x = np.array([[1, 2, 3]]); → x.ndim=2, x.shape=(1,3)
  - funkcije np.zeros, np.ones, np.arrange, np.linspace, np.empty, …
- indeksiranje i izrezivanje je isto kao kod lista.

NumPy, napredno indeksiranje (fancy indexing)

- Indeksiranje pomoću listi, NumPy polja i sl. sa **cjelobrojnim** elementima. Npr:
  - x = np.array([[1, 2, 3, 4], [40, 50, 60, 70], [-10, -20, -30, -40]]) →
    x = array([[1, 2, 3, 4],
    [40, 50, 60, 70],
    [-10, -20, -30, -40]])
  - x[:, [0, 2, 2]] → x = array([[1, 3, 3],
    [40, 60, 60],
    [-10, -30, -30]]).
- Nap.: int se **ne može** zamijeniti s float (u koje su uključeni nan i inf). Npr.:
  - x = np.array([1, 2, 3]) → cjelobrojno polje
    x[1] = 3.5 → javlja pogrešku
    x = x.astype(np.float) → mijenja tip u float
    x[1] = np.nan → x = array([1., nan, 3.]).

NumPy, napredno indeksiranje, nast.

- Operator np.r_ vrši sljepljivanje polja te indeksa (slice) uzduž prve osi (po **retcima**). Npr.:
  - np.r_[np.array([3, 5 ,-2]), [-1, 2], np.nan] →
    array([3., 5., -2., -1., 2., nan])
  - b = np.r_[2:6, 7:3:-1] → array([2, 3, 4, 5, 7, 6, 5, 4])
- Operator np.c_ vrši sljepljivanje polja te indeksa (slice) uzduž druge osi (po **stupcima**). Npr.:
  - np.c_[np.array([3, 5 ,-2]), [-1, 2, np.nan]] →
    array([[3., -1.],
    [5., 2.],
    [-2., nan]])
  - b = np.c_[2:6, 7:3:-1] →
    array([[2, 7],
    [3, 6],
    [4, 5],
    [5, 4]])
- Indeksiranje pomoću **logičkih izraza** (*boolean indexing*). Npr.:
  - x = np.array([1, 2, 3, 4, 5])
    y = x>2 → y = array([False, False, True, True, True])
    z = x[y] → z = array([3, 4, 5]).

NumPy, mijenjanje oblika polja

- np.reshape, npr.
  - x = np.array([[1, 2, 3, 4], [40, 50, 60, 70], [-10, -20, -30, -40]]) →
    x = array([[1, 2, 3, 4],
    [40, 50, 60, 70],
    [-10, -20, -30, -40]])

    y = x.reshape((4,3)) → array([[1, 2, 3],
    [4, 40, 50],
    [60, 70, -10],
    [-20, -30, -40]])
  - Reshape kad god može vraća pogled (*view*)
    x[-1,-1] = 999 → y = array([[1, 2, 3],
    [4, 40, 50],
    [60, 70, -10],
    [-20, -30, 999]])
  - y = x.copy().reshape(4,3) daje **novo** polje y.

NumPy, mijenjanje oblika polja

- Vektorizirani izrazi:
  - aritmetičke operacije na np.array poljima rade po elementima,
  - funkcije rade po elementima.
- Logički i uvjetni izrazi:
  - u numeričkim izrazima, boolean **False** se prevodi **u nulu**, a boolean **True** se prevodi u 1.
- np.where, np.nonzero su ekvivalent od find u Matlabu. Npr.:
  - x = np.array([1, 2, 3, 4]) → array([1, 2, 3, 4])
    x//2*2==x → array([False, True, False, True])
    (x//2*2==x).nonzero() → (array([1, 3]),)

NumPy, matrične i vektorske operacije

- @ -> **matrično** množenje np polja (isto i np.matmul; može i s np.dot)
  - C = A @ B → C = A*B (matrično), A i B su tipa np.ndarray,
    C = A @ x → C = A*x (matrično), x je vektor .
  - Npr.:

```
A = np.array([[1, 2, 3, 4], [40, 50, 60, 70], [-10, -20, -30, -40]])
        →        A = array([[  1,   2,   3,   4],
                            [ 40,  50,  60,  70],
                            [-10, -20, -30, -40]])
x = np.array([np.arange(1,5)])      →     x = array([[1, 2, 3, 4]])

y = A @ x                           →     javlja pogrešku:

ValueError                          Traceback (most recent call last)
 ----> 1 y = A@x
```

  - ValueError: matmul: Input operand 1 has a mismatch in its core dimension 0, with gufunc signature (n?,k),(k,m?)->(n?,m?) (size 1 is different from 4)

NumPy, matrične i vektorske operacije, nast.

- y = A @ x.T → y = array([[30], je OK.
  [600],
  [-300]])
- Ponašanje ovisi o oblicima (shape) ulaznih argumenata!! -> np.matmul?
- np.inner -> skalarni produkt
- Općenito, do uvođenja operatora @, množenje matrica pomocu .dot je bilo malo nespretno, npr. A*B*C bi bilo

  np.dot(A, np.dot(B, C)) ili np.dot(np.dot(A, B), C), a bolji način je:

  A.dot(B.dot(C)) ili A.dot(B).dot(C)
- postoji i tip matrix, ali se upotreba ne ohrabruje!

NumPy, računanje s poljima različitih oblika - emitiranje (broadcasting)

Operacije među poljima su OK ako su polja istog oblika ili je jedno polje skalar.

- ```
  >>> from numpy import array
  >>> a = array([1.0, 2.0, 3.0])
  >>> b = array([2.0, 2.0, 2.0])
  >>> a * b
  array([ 2.,  4.,  6.])
  ```
- ```
  >>> from numpy import array
  >>> a = array([1.0,2.0,3.0])
  >>> b = 2.0
  >>> a * b
  array([ 2.,  4.,  6.])
  ```
- U drugom slučaju je skalar b emitiran po polju a.
- Moćno I elegantno poopćenje → emitiranje (broadcasting)
- Opći postupak (pravilo)
  - Uspoređuju se oblici (shape) počevši od **kraja**
  - Emitiranje kreće ako su pripadne dimenzije **jednake** ili je neka **1**
  - Nap.: Nije nužno da broj dimenzija bude jednak!
  - Oprez: Ako nismo svjesni, do emitiranja može doći i kada to ne želimo!

NumPy, računanje s poljima različitih oblika - emitiranje (broadcasting), nast.

- Detaljnije:
  - https://numpy.org/doc/stable/user/theory.broadcasting.html
  - https://numpy.org/doc/stable/user/basics.broadcasting.html

MatPlotLib

- Praktički svaki račun završava slikom (grafikom).
- **MatPlotLib** je opća Py biblioteka za proizvodnju 2D i 3D grafova visoke kvalitete. Postoje i druge specijalizirane biblioteke.
- Dizajnirana po uzoru na Matlab-ov grafički sustav.

- import matplotlib as mpl
- import matplotlib.pyplot as plt

- Grafika se gradi hijerarhijski:
  - **Slika** (X-prozor ili MS-prozor) je predstavljena objektom tipa **figure**.
  - Pojedini **graf** je predstavljen objektom tipa **axes** koji je metoda od figure.
  - Daljna svojstva (**linije, tekst**, ...) su objekti koji su metode od axes, itd.

MatPlotLib

- Za interaktivno crtanje treba zadati plt.ion() ! BITNO
- fig1, ax = plt.subplots() # stvara objekte figure (fig1) i axes (ax),
- fig1, ax = plt.subplots(3,2) # stvara objekte figure (fig1) i polje objekata axes (ax) oblika (3x2).
- fig1, ax = plt.subplots(nrows=3,ncols=2) # isto kao prethodno; # objekti ax se nalaze medju # atributima of fig1.
- Nap.: Treba razlikovati plt.subplots() - proizvodi novi Figure i skup Subplot-a, od plt.subplot() - dodaje novi Subplot na postojeći Figure
- razne vrste grafova (npr. **plot, step, bar, hist, ...**) su **metode** od ax.

MatPlotLib

- Crtanje linija:
  - lin1 = ax.plot(x,y,'opis_lin')
  - lin1 = ax.plot(x1,y1,'opis_lin1', x2,y2,'opis_lin2', …)
  - lin1 = ax.plot(x,Y)
  - lin1 = ax.plot(X,y)
- Upravljanje
  - ax.hold() # Prebacuje iz **on** u **off** i obratno
  - ax.hold(True) # Podrazumijevana vrijednost (default)
  - ax.hold(False)
  - ax.cla() # Briše sve iz aktivnog axis-a ax, preferirani način
  - plt.close() # Zatvara (briše) aktivnu sliku
  - plt. close(fig1) # Zatvara sliku (objekt) fig1
  - plt.close('all') # Zatvara sve slike

MatPlotLib

- metode za uredjenje **grafa**:
  - ax.set_title('naslov')
    ax.set_xlabel('opis_x_osi')
    ax.set_ylabel('opis_y_osi')
    ax.grid() # dodaje grid prema tickovima
    ax.text(x0,y0, 'tekst', svojstva) # ispisuje ‘tekst’ na položaju (x0,y0)
- metode (osnovni atributi) **linije**:
  - **color**, **linewidth** (ili lw), **linestyles** (ili ls), **marker**.
  - Mogu se zadati kod poziva funkcije plot, kao imenovani argumenti.
  - Radi i opis linije poput '**go--**' (zeleno, marker 'o', linija crtkana).
- **legenda** se zadaje metodom axes-a:
  - ax.legend() # samo linije s label='opis'
    ax.legend(labels) # postojeće linije redom
    ax.legend((line1, line2, line3), ('label1', 'label2', 'label3'))
    # samo zadane linije, a'la Matlab

MatPlotLib

- Crtanje izolinija (kontura)
  - ax.contour(z) # crta konture u polju z tipa (m,n)
  - ax.contour(x,y,z) # x i y su koordinate točaka u z; 2D polja istog oblika (shape) kao z; nivoi se određuju automatski
  - ax.contour(x,y,z,v) # ako je v broj (*int*) on zadaje broj izolinija; ako je v polje (vektor) onda zadaje nivoe
  - cc = ax.contour(...) # vraća odgovarajući objekt
  - cc.clabel() # označava izolinije (najvažnija metoda)
  - cc.clabel(fmt='%1.2f',fontsize=9, …) # specifična svojstva labela
  - ax.contourf(…) # radi kao i contour, ali tako da oboji područja između izolinija
  - fig.colobar(cc) # sa strane dodaje skalu u boji
- Spremanje slike u datoteku
  - plt.savefig('ime.png')
  - plt.savefig('ime.tif')

Literatura

- Z. Kalafatić i dr. (2016): Python za znatiželjne, Element, 513 str.
- R. Johansson (2015): Numerical Python, A Practical Techniques Approach for Industry, Apress, 487 str.
- https://www.srce.unizg.hr/files/srce/docs/edu/osnovni-tecajevi/d105_polaznik.pdf

- https://www.python-course.eu/why_python.php
- https://www.scipy-lectures.org/_downloads/ScipyLectures-simple.pdf

- Prmjeri s kodovima za crtanje:
  - https://matplotlib.org/2.2.3/gallery/index.html
  - https://matplotlib.org/3.0.0/gallery/index.html