

UVOD U RAČUNALNO RAZMIŠLJANJE

1. Što je kodiranje, a što programiranje?

- 1.1. Kodiranje: pisanje programskog koda na osnovu prethodno dobivenih instrukcija, odnosno zahtjeva. Kodiranje je dio programiranja.
- 1.2. Programiranje: širi pojam koji obuhvaća kodiranje, ali i razradu instrukcija, odnosno predstavlja razradu svih potrebnih koraka koji će dovesti do rješenja problema.
- 1.3. Instrukcija: skup naredbi poredanih točno definiranim redoslijedom koje računalo "slijepo" slijedi.
- 1.4. Naredba je najmanji dio programskog jezika koja predstavlja određenu radnju koju treba izvršiti

2. Računalno razmišljanje

- 2.1. Računalno razmišljanje je način rješavanja problema koji se može primijeniti na rješavanje problema iz života, ne samo problema povezanih s računarstvom.
- 2.2. To je misaoni proces tijekom kojeg definiramo problem te njegove manje dijelove na način da se rješenje može opisati kao slijed jednoznačno definiranih koraka.
- 2.3. Računalnim razmišljanjem možemo doći do rješenja za kompleksni problem te isti predstaviti na način kojeg mogu razumjeti osobe, računala ili oboje.
- 2.4. Računalno razmišljanje NIJE razmišljanje kao računalo ili robot, nije programiranje






3. Računalno razmišljanje

- 3.1. Ključni elementi:
 - 3.1.1. *Dekompozicija - rastavi problem na manje djelove.*
 - 3.1.2. *Uočavanje uzoraka - uočiti ponavljanja i sličnosti.*
 - 3.1.3. *Apstrakcija - identifikacija i fokus samo na važne elemente rješenja, služi za kreiranje modela*
 - 3.1.4. *Algoritam - uputa korak po korak kako doći do rješenja, prikaz kao dijagram toka ili pseudokod*

- 3.2. Provjera kvalitete rješenja: ne ulazi u osnovne elemente, nužan korak za kvalitetno i ispravno rješenje problema
 - 3.2.1. *uspješan algoritam: razumljivost (jasnoća), kompletnost (uključuje cijeli problem?), efikasnost (složenost)*

4. Dijagram toka

- 4.1. Jedan od najčešće korištenih načina za grafički prikaz algoritma
- 4.2. Pomoću simbola definira tip koraka i redoslijed njihova izvršavanja
- 4.3. Sastavni elementi:
 - 4.3.1. *početak / kraj - prvi, odnosno zadnji korak*
 - 4.3.2. *proces - instrukcija, zadaća, aktivnost koju treba izvršiti*
 - 4.3.3. *odluka - uvjetno grananje slijeda aktivnosti*
 - 4.3.4. *ulaz / izlaz podataka - ulaz su informacije čijom obradom se kreiraju izlazne informacije*
 - 4.3.5. *smjer - povezuje simbole u cjelinu, definira slijed izvršavanja koraka algoritma*

	Start/end
	Arrows
	Input/Output
	Process
	Decision

5. Pseudo kod

- 5.1. Programski kod koji koristi slobodan jezik kako bi opisao neki problem ili algoritam.
- 5.2. Nije vezan za određeni programski jezik.
- 5.3. Pseudo kôd može sličiti kôdu nekog programskog jezika, ali pseudo kôd nema definiranu sintaksu.
- 5.4. Važno je zapamtiti da se programski kôd piše po točno definiranim pravilima i može se pokretati i izvršavati na računalu, dok pseudo kôd nema nikakva pravila.

Algorithm 1 Quicksort

```
1: procedure QUICKSORT( $A, p, r$ )
2:   if  $p < r$  then
3:      $q = \text{PARTITION}(A, p, r)$ 
4:     QUICKSORT( $A, p, q - 1$ )
5:     QUICKSORT( $A, q + 1, r$ )
6:   end if
7: end procedure
8: procedure PARTITION( $A, p, r$ )
9:    $x = A[r]$ 
10:   $i = p - 1$ 
11:  for  $j = p$  to  $r - 1$  do
12:    if  $A[j] < x$  then
13:       $i = i + 1$ 
14:      exchange  $A[i]$  with  $A[j]$ 
15:    end if
16:  exchange  $A[i]$  with  $A[r]$ 
17: end for
18: end procedure
```

OSNOVE PROGRAMIRANJA U PYTHON-U

1. O Python-u

- 1.1. Python je programski jezik opće namjene.
- 1.2. Nastao je krajem 1980-tih, prva javna verzija (0.9.0) objavljena je 1991, verzija 1.0 objavljena je 1994, verzija 2.0 objavljena je 2000, verzija 3.0 objavljena je 2008.
- 1.3. Osmislio ga je programer Guido van Rossum u Nizozemskoj.
- 1.4. Ime Python proizašlo je iz televizijske serije Monty Python's Flying Circus.

2. PEP

- 2.1. Python Enhancement Proposals
- 2.2. skup dokumenta koji definiraju dizajn, pravila i konvenciju pisanja Python koda napisanih za Python korisnike.
- 2.3. Postoje tri različita tipa:
 - 2.3.1. *Standards* - opisuju nove značajke ili implementacije u Pythonu
 - 2.3.2. *Informational* - smjernice i informacije za korisnike
 - 2.3.3. *Process* - objašnjenje procesa u Pythonu
- 2.4. PEP 20 - The Zen of Python: 1999. Tim Peters na Python mailing listu za unapređenje Python programskog jezika poslao je skup od 19 načela kako napisati dobar softver, uključena u PEP 2004.

3. Variable

- 3.1. Varijable - skup podataka korištenih u programskom kodu, predstavljaju "human readable" nazive dijelova memorije računala (mjesto gdje se informacija tj. podatak čuva tijekom izvođenja koda)
- 3.2. Što čini varijablu?
 - 3.2.1. *Tip* (npr. *String*, *Integer*, ...)
 - 3.2.2. *Podatak* (npr. stvarna vrijednost "neki tekst", 5, ...)
 - 3.2.3. *Ime* (naziv varijable definiran od osobe koja razvija kod)
- 3.3. Naziv varijable ne smije biti ključna riječ, početi brojkom, imati razmak, imati posebne znakove: !?@#\$\$%...
- 3.4. Nazivi varijabli u Pythonu su case sensitive - razlikuje velika i mala slova u nazivu varijable
- 3.5. PEP 8: Style Guide for Python Code > naming conventions
- 3.6. Ključne riječi (python > help() > keywords: 35 komada)

not	try	if	class	nonlocal	is
and	except	elif	def	global	in
or	finally	else	lambda	import	as
False	break	for	return	async	assert
True	continue	while	yield	await	tdel
None	pass	with	from	raise	., : () [] { }

- 3.7. Preporuke za imenovanje varijabli. Naziv varijable bi trebao:
 - 3.7.1. opisivati podatak koji je pohranjen u varijabli
 - 3.7.2. ili imati sve znakove napisane malim slovima i znak "_" (underscore) umjesto razmaka - snake_case
 - 3.7.3. ili biti bez razmaka, tako da svaka riječ počinje velikim slovom - camelCase
 - 3.7.4. biti bez (hrvatskih) dijakritičkih znakova
 - 3.7.5. biti dosljedan - varijabla je dio programskog kôda > bolje koristiti engleski jezik, nikako ne miješati jezike

- 3.8. Komentari u kodu - dio programskog koda koji se ne izvršava
- 3.8.1. *komentari u jednoj liniji počinju znakom #*
- 3.8.2. *komentari na više linija počinju i završavaju 3x dvostrukim navodnicima (""" ... """)*

4. Tipovi podataka

- 4.1. Osnovni tipovi podataka u Python-u
- 4.1.1. *numerics (brojevi), sequences (kolekcije), mappings (mapiranja), classes (klase), instances (instance), exceptions (greške)*
- 4.2. Numerics (brojevi):
 - 4.2.1. *Integer (int) - cijeli brojevi [broj: int = 12]*
 - 4.2.2. *Boolean (bool) - True/False [istina: bool = True] - subtype of integers*
 - 4.2.3. *Float (float) - decimalni brojevi [decimalniBroj: float = 123.4]*
 - 4.2.4. *Complex (complex) - kompleksni brojevi [kompleksniBroj: complex = 123.4+123.4j]*
 - 4.2.5. *dodatno kroz module: Decimal, Fraction*
- 4.3. Sequences (kolekcije)
 - 4.3.1. *Objekti koji mogu poprimiti više različitih objekta i spremiti ih kao cjelinu.*
 - 4.3.2. *Omogućuju dohvat i iteraciju kroz objekte da bi im mogli pristupiti.*
 - 4.3.3. *Indeksi indeksiranih kolekcija (lista, tuple) kreću od 0 i jedinstveni su unutar kolekcije*
 - 4.3.4. *Lista: Definira se preko uglatih zagrada [] ili pozivom konstruktora list(). Promjenjiv tip podatka*
 - 4.3.5. *String - tekst, piše se unutar ' ili " navodnika, nepromjenjiv tip podatka [tekst: str = 'Ovo je tekst']*
 - 4.3.6. *Tuple: Definira se preko () zagrada. Nepromjenjiv tip podatka.*
 - 4.3.7. *Set: Definira se preko { } zagrada ili pozivom konstruktora set()*
 - *set, za razliku od liste, odbija duplikate (ne može spremiti iste vrijednosti ili ih obriše ako je kreiran iz liste)*
- 4.4. Mappings (mapiranja)
- 4.5. Objekti koji mapiraju hashable vrijednosti na objekte. Promjenjivi tip objekta.
- 4.5.1. *Dictionary: Trenutno jedini standardni mapping. Definira se preko { } zagrada ili pozivom konstruktora dict()*
 - *dict i set definiraju se preko { }, set sprema vrijednosti preko indexa (kao lista, tuple), dict koristi key-value.*
- 4.6. Konverzije tipova podataka
- 4.7. konstruktori: int(var), float(var), complex(var), str(var), bool(var), ...
- 4.7.1. *ASCII translator: chr(int_var) > u znak, ord(str_var) > u broj*

5. Greške u programskom kodu

- 5.1. BUG: buba u računalu zbog koje je prestalo raditi, zbog koje se kolokvijalno greške u kodu nazivaju bug
- 5.2. Tipovi grešaka
 - 5.2.1. *Syntax Error - krivo napisane naredbe, nazivi funkcija, najčešće IDE točno ukazuje gdje se nalazi greška*
 - 5.2.2. *Runtime Error / Exception - greške prilikom izvođenja programa*
 - 5.2.3. *Bugs - greške koje predstavljaju krivo funkcioniranje programa - radi, ali ne ono što bi trebao*
 - 5.2.4. *mnoge druge specifične vrste grešaka: ArithmeticError (Overflow, ZeroDivision, FloatingPoint), BufferError, LookupError (Index, Key), EOFError, ImportError (ModuleNotFound), MemoryError, NameError, NotImplementedError, OSError, RecursionError, ReferenceError, StopIteration, SystemError, TypeError, ValueError, EnvironmentError, IOError, ConnectionError, PermissionError, TimeoutError, ...*

6. Osnove sintakse:

6.1. Grupiranje instrukcija u blokove:

6.1.1. Python za grupiranje koda umjesto {} zagrada koristi uvlaku (tab) ili 4 razmaka

6.1.2. jedan od načina kako Python želi povećati čitljivost programskog koda (kao paragraf)

6.2. for petlja: ponavlja neku instrukciju određeni broj puta

- *for element in kolekcija:*
- *aktivnost nad element*

6.3. while petlja: ponavlja neku instrukciju do (ne)ispunjenja uvjeta

- *while uvjet:*
- *izvršavanje bloka instrukcija*

6.4. uvjetno izvršavanje: utjecanje na tijek petlje i drugih struktura

6.4.1. *break: kad želimo izvršavanje petlje završiti prije kraja ("nasilno" izaći iz petlje)*

6.4.2. *continue: kad ne želimo izaći iz petlje nego želimo da se neki ciklusi preskoče ako su ispunjeni neki uvjeti*

6.4.3. *pass: kad trebamo izvršiti naredbu koja ne radi ništa*

6.5. uvjetno izvršavanje: if grananje

- *if uvjet: ...*
- *elif uvjet: ...*
- *else: ...*

6.6. Funkcije - skup instrukcija koje su izdvojene u zasebnu cjelinu

- *reusing - jednom napisani kod možemo koristiti koliko god puta nam treba*
- *održavanje - olakšano, jednostavnije pregledavanje koda*
- *debugging - jednostavniji jer nema redundancije (ponavljanja)*

6.6.1. *funkcija bi trebala raditi samo jednu aktivnost*

6.6.2. *pokrećemo ih pomoću poziva funkcije (call)*

6.6.3. *funkcije imaju:*

- *naziv: koristi se za pokretanje funkcije, ista pravila kao za varijable*
- *argumente ili parametre: podaci koje će funkcija prihvatiti na ulazu i obraditi*
- *tijelo ili blok instrukcija: skup instrukcija koje će izvršiti određenu akciju*
- *rezultat ili return parametar: može ili ne mora vratiti nekakav podatak*

6.6.4. *moгу biti ugrađene (built-in), neke treba uključiti (import modula)*

6.6.5. *korisnički definirane funkcije:*

- *def naziv_funkcije(argumenti):*
- *""" docstring (opcionalno) """*
- *blok instrukcija*

6.6.6. *osnovne ugrađene funkcije:*

- *A*
- *abs(var) - vraća apsolutnu vrijednost integera ili floata, odnosno modul kompleksnog broja*
- *all(iterable) - vraća True ako su svi elementi kolekcije true ili ako je kolekcija prazna*
- *any(iterable) - vraća True ako je bilo koji element kolekcije true, False ako je kolekcija prazna*
- *C*
- *callable(object) - vraća True ako je objekt iz vrste callable (funkcija ili klasa s __call__() metodom)*
- *D*
- *delattr(object,name) - briše imenovani atribut klase (delattr(inst,attr) = del inst.attr)*
- *dir(object) - lista naziva u lokalnom obuhvatu, ako je klasa s oo __dir__(), daje listu atributa*
- *E*
- *enumerate(iterable,start=0) - vraća tuple s indeksom i elementom kolekcije (indeks,element)*
- *eval(expression,globals=None,locals=None) - parsing i evaluacija Python naredbe (tj. liste uvjeta)*
- *exec(object,globals=None,locals=None/,*,closure=None) - omogućuje dinamičko izvršavanje koda*

- **F**
- *filter(function,iterable)* - konstruira iterator od elemenata za koje (bool) funkcija vrati True
- *format(value,format_spec="")* - konvertira value u formatiranu reprezentaciju kontroliranu s format_spec
- **G**
- *getattr(object,name,default)* - vraća vrijednost atributa objekta (*getattr(inst,attr) = inst.attr*)
- **H**
- *hasattr(object,name)* - vraća True ako objekt ima atribut, False ako nema
- *help(objekt)* - daje osnovne podatke o objektu (modul, funkcija, klasa, metoda, keyword, doc. topic)
- *help()* - pali se interaktivni help, može se koristiti i za vlastite funkcije (čita docstring)
- **I**
- *id(object)* - vraća "identitet" objekta - jedinstveni broj konstantan za objekt tijekom njegovog trajanja
- *input(prompt=' ', /)*, *prompt* = poruka prije inputa, *printa* se bez newline charactera
- *isinstance(object,classinfo)* - vraća True ako je objekt instanca klase ili jedne od klasa ako se da tuple
- *issubclass(class,classinfo)* - vraća True ako je klasa potklasa nadklase ili jedne od nadklasa iz tuplea
- *iter(object)* - vraća iterator kolekcije koja podržava iterable (*__iter__()*) ili sequence (*__getitem__()*) protokol
- **L**
- *len(naziv_kolekcije)* - vraća broj elemenata pohranjenih u kolekciji, odnosno sekvenci
- **M**
- *map(function,iterable,*iterables)* - primjenjuje funkciju na svaki element kolekcije(a)
- *min(iterable,key=None)* - vraća najmanji element iz kolekcije, alternativno *min(arg1,arg2,*args,key=None)*
- *max(iterable,key=None)* - vraća najveći element iz kolekcije, alternativno *max(arg1,arg2,*args,key=None)*
- **N**
- *next(iterator,default)* - vraća slijedeći element iteratora, zove *__next__()* metodu
- **O**
- *open(file,mode='r',buffering=-1,encoding=None,errors=None,newline=None,closefd=True,opener=None)*
- **P**
- *print(*args,sep=' ', end='\n', file=sys.stdout, flush=False)* - prikaz na ekranu
- **args* = output, često se koriste f-stringovi zbog fleksibilnosti "... {var} ... {var}...", mogu i klasični
- *sep* = separator, *end* = zadnji znak u retku, *file* = mjesto ispisa, *flush* = da li prazni stream
- *property(fget=None,fset=None,fdel=None,doc=None)* - vraća property atribut, koristi se za managed attr
- kada se specificira kao decorator, ima *@property > @p.getter + @p.setter + @p.deleter*
- **R**
- *range(start,stop,step)* - iterator (neopadajućeg/nerastućeg) niza cijelih brojeva, često se koristi u petljama
- *range(n)*: 0-n, korak:1, *range(n,m)*: n-m, korak:1, *range(n,m,k)*: n-m, korak:k
- *repr(object)* - vraća string sa printable reprezentacijom, u klasi se oo pomoću *__repr__()* metode
- *reversed(seq)* - vraća obrnuti iterator, seq sadržava *__reversed__()* metodu ili podržava sequence protocol
- **S**
- *setattr(object,name,value)* - ažurira vrijednost atributa (*setattr(inst,attr,value) = inst.attr=value*)
- *slice(start,stop,step)* - vraća slice sekvence koji čini set indeksa specificiranih prema *range(start,stop,step)*
- *sorted(iterable,/,*,key=None,reverse=False)* - vraća sortiranu listu elemenata kolekcije (Tim sort)
- *sum(iterable,/,start=0)* - zbraja start i elemente kolekcije (obično brojevi), vraća zbroj
- **V**
- *vars(object)* - vraća *__dict__* atribut za objekt koji ga ima (modul, klasa, instanca ili slično)
- **Z**
- *zip(*iterables,strict=False)* - iterira preko više kolekcija paralelno, vraća tuple po tuple redova
- *strict=True* - zahtijeva da kolekcije budu jednake duljine, inače završava s najkraćim

- 6.7. Numerics (bool, int, float, complex)
- 6.7.1. osnovne naredbe (za kompleksne ne vrijede one koje nisu primjenjive)
- uređaj: <, <=, ==, >=, >, !=, is (identitet), is not (negativ identiteta)
 - operatori: x+y, x-y, x*y, x/y, x//y, x%y, -x, +x, abs(x), divmod(x,y)=(x//y,x%y), pow(x,y)=x**y, round(x,n)
 - bitwise operatori: OR x|y, XOR x^y, AND x&y, SHIFT LEFT x<<y, SHIFT RIGHT x>>y, INVERT ~x
- 6.7.2. konverzije brojevnih sustava:
- bin(cijeli_broj), oct(cijeli_broj), hex(cijeli_broj)
 - konstruktor: int("cijeli_broj", baza, baza = 2,8,10,16,...)
- 6.8. String - kolekcija znakova
- 6.8.1. escape characters: \ = \, \' = ', \" = ", \n = novi red, \t = tab, \r = carriage return, \b = backspace ...
- 6.8.2. zapravo kolekcija znakova, može se s njim raditi kao s listom, osim što je nepromjenjiv
- 6.8.3. osnovne naredbe
- count(sub), find(sub), index(sub), join()
 - capitalize(), swapcase(), title(), upper(), lower(), format(*args,*kwargs), zfill(width)
 - isalnum(), isalpha(), isdecimal(), isdigit(), islower(), isnumeric(), isspace(), istitle()
 - lstrip(chars), rstrip(chars), strip(chars), split(sep,maxsplit), splitlines()
 - startswith(prefix), removeprefix(prefix), endswith(suffix), removesuffix(suffix), replace()
- 6.9. Lista - kolekcija niz podataka
- 6.9.1. osnovne naredbe:
- naziv_liste[indeks] = nova_vrijednost - izmjena vrijednosti pohranjene u element na poziciji indeks
 - slicing: naziv_liste[start:stop:step] - način kreiranja nove liste na osnovu manipulacije postojećom
 - naziv_liste.append(novi_element) - dodavanje novog elementa na kraj liste
 - naziv_liste.clear() - naredba za brisanje svih elemenata liste
 - naziv_liste.copy() - naredba za kopiranje liste (shallow copy, ne kopira zapravo elemente)
 - naziv_liste.count(element) - prebrojava koliko se puta element pojavljuje u listi
 - naziv_liste.extend(nova_lista) - proširuje postojeću listu novom listom
 - naziv_liste.index(element) - dohvaća prvi indeks pozicije na kojoj se nalazi element
 - naziv_liste.insert(indeks,element) - umetanje elementa točno ispred pozicije označene indeksom
 - naziv_liste.pop(indeks) - uklanjanje elementa na poziciji označenoj indeksom, default zadnji element
 - naziv_liste.reverse() - naredba za obrtanje redoslijeda elemenata liste
 - naziv_liste.sort() - naredba za sortiranje elemenata (koristi Tim sort)
- 6.10. Dictionary - mapiranje parova podataka
- 6.10.1. svaki element ima dva dijela: key (ključ), value (vrijednost)
- 6.10.2. key: mora biti jedinstven, dopušteni tipovi podataka su string, brojevi i tuple
- 6.10.3. pomoću keya pristupamo drugom dijelu para, podacima (values)
- 6.10.4. value: sadržaj koji želimo pohraniti u kolekciju, može biti bilo koji tip podatka, ili kolekcija
- 6.10.5. osnovne naredbe:
- naziv_dict[key] - dohvaća value pod tim keyem ako postoji
 - naziv_dict.has_key(key) ; if key in dict ; if dict.get(key) == None - provjerava postoji li key
 - naziv_dict.clear() - briše sve key-value parove iz dictionary-ja
 - naziv_dict.copy() - radi shallow copy cijelog dictionary-ja
 - naziv_dict.fromkeys(keys_iterable,v) - kreira dictionary sa specificiranim keyevima sa vrijednošću v
 - naziv_dict.get(key) - vraća vrijednost za specificirani key
 - naziv_dict.items() - dohvaća key-value parove kao listu tupleova
 - naziv_dict.keys(), naziv_dict.values() - dohvaća keys / values kao listu
 - naziv_dict.pop(key,default) - briše key-value par pod keyem
 - naziv_dict.popitem() - briše item koja je zadnja dodana u dict
 - naziv_dict.setdefault(k,v) - vraća vrijednost za specificirani key, ako nema, dodaje key-value par
 - naziv_dict.update(d) - dopunjava dictionary specificiranim key-value parovima

- 6.11. tuple (N-terac)
- 6.11.1. često se koristi kao tip pohrane povezanih podataka unutar neke kolekcije
- `naziv_tuple.count()` - prebrojava koliko se puta element pojavljuje u tuple-u
 - `naziv_tuple.index(element)` - dohvaća prvi indeks pozicije na kojoj se nalazi element
- 6.12. try-except-else-finally:
- 6.12.1. try: blok koda koji želimo izvršiti
- 6.12.2. except: blok koda koji se izvršava ako se dogodi neka iznimka, može biti više except blokova
- 6.12.3. else: blok koda koji se izvršava ako se try blok uspije izvršiti
- 6.12.4. finally: blok koda koji će se izvršiti neovisno je li se dogodila greška, najčešće za otpuštanje vanjskog resursa
- try:
 - blok naredbi
 - except `someError` as `se`:
 - blok naredbi
 - except `otherError` as `oe`:
 - blok naredbi
 - else:
 - blok naredbi
 - finally:
 - blok naredbi

7. Python moduli

- 7.1. omogućuju odvajanje funkcionalnosti u zasebne cijeline i datoteke - može biti datoteka ili direktorij
- 7.2. postoje ugrađeni moduli (dolaze s Pythonom), možemo kreirati mi, a mogu biti i skinuti s interneta (npr. pip)
- 7.3. module koristimo kako bismo imali bolje organiziran kod koji možemo lakše ispravljati, proširiti, debugirati i lakše se snalaziti u istom.
- 7.4. paketi: modul je svaka .py datoteka, paket je više međusobno povezanih .py datoteka (modula), smještenih u jednu mapu, koji zajedno čine jednu smislenu cjelinu
- 7.4.1. svaki paket unutar mape mora imati jednu `__init__.py` datoteku, koja može biti i prazna
- 7.4.2. kolokvijalno se i dalje nazivaju moduli, iako se razlikuju od onog što se definira kao modul
- 7.5. osnovni ugrađeni moduli (<https://docs.python.org/3/py-modindex.html>, za rad je potreban import)
- 7.5.1. random - funkcije za generiranje nasumičnih brojeva
- `randint(start,stop)` - vraća nasumični broj unutar određenog raspona
 - `choice(iterable)` - vraća nasumični element iz kolekcije (`string`, `range`, `list`, `tuple`, ...)
 - `choices(iterable,weights,k)` - vraća kolekciju duljine `k` uz definirane vjerojatnosti članova osnovne kolekcije
 - `shuffle(iterable)` - vraća kolekciju nasumično presloženu
 - `sample(iterable,k)` - vraća podniz duljine `k` iz osnovne kolekcije (bez ponavljanja, `k<=len(iterable)`)
 - `random()` - vraća nasumični float između 0.0 i 1.0
 - `...variate()` - vraća nasumični float iz određene distribucije (eksponencijalna, normalna, gamma, ...)
- 7.5.2. math - matematičke funkcije i konstante
- `sin()`, `asin()`, `cos()`, `acos()`, `tan()`, `atan()` - trigonometrijske funkcije
 - `sinh()`, `asinh()`, `cosh()`, `acosh()`, `tanh()`, `atanh()` - hiperboličke funkcije
 - `pow()` - potencija, `exp()`, `log()` - eksponencijalna i logaritamska funkcija
 - `comb()` - binomni koeficijent, `perm()` - broj permutacija
 - `erf()`, `erfc()` - error i komplementarna error funkcija
 - `dist()` - euklidska udaljenost, `hypot()` - euklidska norma, `isclose()` - jesu li dvije vrijednosti blizu
 - `floor()`, `ceil()` - zaokružuje na niži / viši integer, `trunc()` - uklanja dio iza dec. točke
 - `factorial()` - faktoriyel, `gamma()` - gamma funkcija, `lgamma()` - log gamma funkcija
 - `gcd()` - najveći zajednički dijelitelj
 - `isfinite()`, `isinf()` - je li broj (bes)konačan
 - konstante: beskonačno, `e`, `NaN`, `pi`, `tau`, ...

- 7.5.3. *os* - funkcije za rad u nekim elementima operativnog sustava
- `_exit()`, `abort()`, `kill()`, `chdir()`, `chmod()`, `chown()`, `chroot()`, `getcwd()`, `getegid()`, `getenv()`, `geteuid()`, `getgid()`, `getpid()`, `getuid()`, `major()`, `minor()`, `mkdir()`, `mkfifo()`, `open()`, `pipe()`, `popen()`, `read()`, `remove()`, `rmdir()`, `setgid()`, `setuid()`, `stat()`, `symlink()`, `umask()`, `wait()`, `write()`, ... linux style
 - `os.path.join(path, dir, dir, ..., file)` - spaja put do file-a
- 7.5.4. *sys* - manipulacija različitim elementima Python runtime environmenta
- `stdin` - standard input ; `stdout` - standard output ; `stderr` - standard error
 - na jednostavan način omogućuje redirekciju - samo izmijeniti vrijednost nekog od gornjih objekata
 - `argv` - lista cmd line argumenata poslanih Python skripti
 - `getrefcount()` - broj referenci na objekt
 - `getsizeof(object)` - veličina objekta u byte-ovima
 - `sys.path[0]` - trenutno aktivni direktorij (iz kojeg je pokrenuta skripta)
 - `platform` - razlikuje "linux" = Linux, "win32" = Windows, "darwin" = macOS
- 7.5.5. *datetime* - funkcije za upravljanje tipom varijabli za pohranu vremena
- `strptime(format)` - konvertira dobiveni datum / vrijeme u određeni format
 - *date* klasa - idealizirani datum prema gregorijanskom kalendaru
 - `year`, `month`, `day` - atributi, `today()` - danas, `replace()`, `isoformat()` - YYYY-MM-DD, `ctime()` > date string
 - `x = datetime.date.today()` - kupi datum za sada
 - `x = datetime.date(2024, 10, 20)` - kreira 2024-10-20
 - `x = datetime.date.fromtimestamp(1326244364)` - kupi Unix timestamp (broj sec od 01.01.1970.)
 - `print(f"godina: {x.year}, mjesec: {x.month}, dan: {x.day}")`
 -
 - *time* klasa - idealizirano vrijeme neovisno o danu
 - `hour...microsecond` - atributi, `replace()`, `isoformat()`
 - `x = datetime.time.now()`
 - `x = time(12, 34, 56, 123456)` - kreira 12:34:56.123456
 - `print(f"sat: {x.hour}, minuta: {x.minute}, sekunda: {x.second}, ms: {x.microsecond}")`
 - *datetime* klasa - kombinacija *date* i *time* klase
 - `year...microsecond` - atributi, `date()`, `today()`, `time()`, `now()`, `replace()`, `timestamp()`, `isoformat()`, `ctime()`
 - `x = datetime.datetime.now()` - kupi datum i vrijeme za sada
 - `x = datetime.datetime(2024, 10, 20, 12, 34, 56, 123456)` - kreira 2024-10-20 12:34:56.123456
 - `print(f"godina: {x.year}, mjesec: {x.month}, dan: {x.day}, sat: {x.hour}, minuta: {x.minute}, ...")`
 - `print(f"godina: {x.strftime('%Y')}, mjesec: {x.strftime('%m')}, dan: {x.strftime('%d')}, ...")`
 -
 - `x = datetime.date.fromisoformat("2024-10-20")` - kreira iz ISO 8601 formata
 - `print(f"ISO format: {x.isoformat(sep='T', timespec='auto')}")` - kreira string u ISO 8601 formatu
 - * YYYY-MM-DDTHH:MM:SS.ffffff, za separator = T, ako mikrosekunda nije 0
 - *timedelta* - trajanje između dva date-time do rezolucije mikrosekunde
 - `days`, `seconds`, `microseconds` - atributi,
 - `t1 = date(...)`, `t2 = date(...)` ; `t1 = time(...)`, `t2 = time(...)` ; `t1 = datetime(...)`, `t2 = datetime(...)`
 - `t3 = t2 - t1` - kreira *timedelta* objekt koji mjeri vremensku udaljenost t1 i t2
 -
 - `td = datetime.timedelta(weeks=2, days=50, hours=5, minutes=5, seconds=27, microseconds=10)`
 - konvertira sve u days + seconds + microseconds
 - `timedelta.total_seconds()` - konvertira sve u sekunde
 -
 - *timezone* - implementacija tzinfo bazne klase - time zone info objekti

7.5.6. time - funkcije vezane za mjerenje vremena

- `gmtime()` - konvertira protekli broj sekundi od "početka vremena" (epoch) u `struct_time`, UTC
- `localtime()` - isto kao `gmtime`, samo konvertira u lokalno vrijeme
- `perf_counter()` - mjeri vremensku udaljenost između dva call-a u max res, system wide
- `process_time()` - vraća zbroj sistemskog i user CPU vremena trenutnog procesa, bez sleep
- `sleep()` - zaustavlja izvršavanje na određeni broj sekundi
- `strftime(format)` - konvertira dobiveno vrijeme u određeni format

%a	abbreviated weekday name	%M	minute as a decimal number [00,59]
%A	full weekday name	%p	locale's equivalent of either AM or PM
%b	abbreviated month name	%S	second as a decimal number [00,61]
%B	full month name	%U	week number (0=Sunday) as a decimal number [00,53]
%c	appropriate date and time representation	%w	weekday as a decimal number [0=Sunday,6]
%d	day of the month as a decimal number [01,31]	%W	week number (0=Monday) as a decimal number [00,53]
%f	microseconds as a decimal number	%x	local appropriate date representation
%H	hour (24-hour clock) as a decimal number [00,23]	%X	local appropriate time representation
%I	hour (12-hour clock) as a decimal number [01,12]	%y	year without century as a decimal number [00,99]
%j	day of the year as a decimal number [001,366]	%Y	year with century as a decimal number.
%m	month as a decimal number [01,12]	%z	TZ offset indicating +/- time difference from UTC/GMT

7.5.7. csv (comma separated value format handling)

- `csv.reader(csvfile, dialect, **)` - čita red po red: with `open(f)` as `f`: `r = csv.reader(f)`, for row in `r ...`
- `csv.writer(csvfile, delimiter, **)` - piše red po red: with `open(f, 'w')` as `f`: `w = csv.writer(f)`, `w.writerow([,])`
- `csv.DictReader(csvfile, dialect...)` - mapira na dict: with `open(f)` as `f`: `dr = csv.DictReader(f)`, for row in `d ...`
- `csv.DictWriter(csvfile, dialect,...)` - mapira iz dict: with `open(f, 'w')` as `f`: `dw = csv.DictWriter(f)`, `dw.writerow({,})`

7.5.8. abc (abstract base classes)

7.5.9. asyncio (asynchronous I/O)

7.5.10. calendar (working with calcs)

7.5.11. collections (container datatypes)

7.5.12. configparser (configuration file parser)

7.5.13. copy (shallow and deep copy operations)

7.5.14. decimal (general decimal arithmetic spec)

7.5.15. email (parsing, manipulating, generating)

7.5.16. enum (enumeration)

7.5.17. fractions (rational numbers)

7.5.18. functools (high-order funcs on callables)

7.5.19. html (helpers and parsers)

7.5.20. importlib (import machinery)

7.5.21. ipaddress (IPv4/v6 manipulation)

7.5.22. io (working with streams)

7.5.23. itertools (iterators for efficient looping)

7.5.24. json (encoding, decoding)

7.5.25. logging (flexible event logger)

7.5.26. multiprocessing (process-based parallelism)

7.5.27. operator (functions corresponding to operators)

7.5.28. pickle (convert py objects to streams)

7.5.29. pprint (pretty print)

7.5.30. re (regex manipulation)

7.5.31. sense_emu (RPi Sense Hat emulator)

7.5.32. shelve (python object persistence)

7.5.33. socket (low-level networking interface)

7.5.34. sqlite3 (DB-API 2.0 implementation)

7.5.35. ssl (TLS/SSL wrapper for sockets)

7.5.36. statistics (math. stat. functions)

7.5.37. struct (interpret bytes as packed binaries)

7.5.38. subprocess (handling and management)

7.5.39. test (regression tests suite)

7.5.40. threading (thread-based parallelism)

7.5.41. tkinter (simple GUI)

7.5.42. types (names for built-in types)

7.5.43. typing (support for type hints)

7.5.44. urllib (requests, parsing, errors)

7.5.45. xml (processing and management)

7.5.46. zlib (gzip compression and decompression)

7.6. bitni vanjski moduli

7.6.1. bs4 (beautiful soup)

7.6.2. jupyter, jupyterlab

7.6.3. matplotlib (plotting)

7.6.4. numpy (numeric)

7.6.5. pillow (PIL image processing)

7.6.6. pandas (dataframes)

7.6.7. paramiko (SSH)

7.6.8. requests (web)

7.6.9. scipy (science)

7.6.10. sqlalchemy (sql orm)

7.6.11. sympy (symbolic python)

7.6.12. statsmodels (stats)

PROGRAMIRANJE U PYTHONU

1. Objektno orijentirano programiranje (OOP)

- 1.1. jedan od pristupa računalnog programiranja.
- 1.2. princip programiranja gdje je rješenje bazirano na skupu objekta koji međusobno komuniciraju i na taj način rješavaju zadan problem.
- 1.3. OOP se ističe:
 - 1.3.1. *jasnim i definiranim arhitekturama aplikacije*
 - 1.3.2. *jasnim i čitljivim kodom*
 - 1.3.3. *malom količinom redundancije*
 - 1.3.4. *velikom količinom reusabilnog koda*

2. Klase i objekti

- 2.1. klasa - korisnički definirani tip podatka kojim se modeliraju objekti sličnih svojstava
 - 2.1.1. *predstavlja predložak, nacrt na temelju kojeg će se definirati varijable unutar programskog koda*
- 2.2. objekt - stvarna instanca neke klase (koja se nalazi u memoriji)
 - 2.2.1. *svaki objekt je definiran stanjem i ponašanjem definiranim u klasi*
- 2.3. sintaksa za kreiranje klase:
 - `class NazivKlase(RoditeljskaKlase):`
 - `""" docstring (opcionalno) """`
 - *karakteristike ...*
 - `def neka_metoda(self):`
 - *blok instrukcija*
 - `def neka_metoda(cls):`
 - *blok instrukcija*
- 2.4. sintaksa za korištenje klase:
 - `objekt = NazivKlase()`
 - `objekt.karakteristika`
 - `objekt.neka_metoda()`
 - `NazivKlase.neka_metoda()`
- 2.5. reference:
 - 2.5.1. *self - odnosi se na instancu nad kojom se metoda/svojstvo poziva*
 - 2.5.2. *cls - odnosi se klasu, za pozivanje class metoda / dohvaćanje svojstava*
- 2.6. naslijeđivanje - jedan od elemenata OOP: klase naslijeđuju svojstva i metode svojih roditelja
 - *super: defines a method and a delegate that expects an action in a subclass*
 - *inheritor: doesn't provide any new names, gets everything defined in super*
 - *replacer: overrides super's method with a version of its own*
 - *extender: customizes super's method by overriding and calling back to run default*
 - *provider: implements the action method expected by super's delegate method*
- 2.7. vrste metoda:
 - *instance method - uzima parametar self, koji je referenca na instancu klase*
 - *class method - uzima parametar cls, koji je referenca na samu klasu*
 - *static method - ne uzima ni self ni cls, opća metoda korisna u klasi*

2.7.1. operator overloading:

<code>__new__</code>	object creation before <code>__init__</code>	<code>__get__</code>	<code>x.attr</code>
<code>__call__</code>	<code>f.calls, x(*pargs,**kargs)</code>	<code>__set__</code>	<code>x.attr=value</code>
<code>__bool__</code>	<code># bool(x), truth tests</code>	<code>__delete__</code>	<code>del x.attr</code>
<code>__len__</code>	<code>len(x)</code>	<code>__getitem__</code>	<code>x[key].x[i:j]</code> , for loops
<code>__del__</code>	object reclamation of x	<code>__setitem__</code>	<code>x[key]=value, x[i:]=iterable</code>
<code>__add__</code>	<code>x + y, x += y (if no __iadd__)</code>	<code>__delitem__</code>	<code>del x[key], del x[i:j]</code>
<code>__iadd__</code>	<code>x += y (or else add)</code>	<code>__lt__, __gt__</code>	<code>x < y, x > y</code>
<code>__or__</code>	<code>x y, x = y (if no __ior__)</code>	<code>__le__, __ge__</code>	<code>x <= y, x >= y</code>
<code>__ior__</code>	<code>x = y (or else or)</code>	<code>__eq__, __ne__</code>	<code>x == y, x != y</code>
<code>__repr__, __str__</code>	<code>print(x), repr(x), str(x)</code>	<code>__iter__, __next__</code>	<code>l=iter(x), next(l); loops, comps,...</code>
<code>__getattr__</code>	<code>x.undefined</code>	<code>__contains__</code>	item in x (any iterable)
<code>__setattr__</code>	<code>x.any = value</code>	<code>__index__</code>	int. value; <code>hex(x), bin(x), oct(x),...</code>
<code>__delattr__</code>	<code>del x.any</code>	<code>__enter__, __exit__</code>	with obj as var

3. Rad s datotekama

- 3.1. Python bez potrebe za instalacijom modula podržava rad s najvažnijim formatima datoteka
- 3.2. za napredan rad sa specifičnim tipovima datoteka potrebno je instalirati specijalizirane module
- 3.3. dok pišemo kod koji koristi vanjske resurse ne smijemo smatrati da ćemo im uvijek imati pristup
- 3.3.1. *možda nemamo pravo pristupa, možda datoteke nema, možda nemamo mrežni pristup serveru/internetu*

3.4. čitanje i pisanje tekstualnih datoteka:

- 3.4.1. 1. otvori datoteku za čitanje / pisanje - "zaključava", onemogućava drugima pristup (otvaranje konekcije)
- 3.4.2. 2. pročitaj sadržaj / zapiši tekst - datoteka mora biti otvorena u pripadajućem modu
- 3.4.3. 3. zatvori datoteku - oslobađa drugim aplikacijama pristup datoteci (zatvaranje konekcije)

3.4.4. korištenjem context managera koraci 1 i 3 su automatizirani

- `with open(file_1_path, [r,w,a]) as naziv_1, open(file_2_path, [r,w,a]) as naziv_2, ...`
- `r = read (default), w = write, a = append`
- konekcija prema datoteci se nakon kraja korištenja automatski zatvara
- ako napravimo `try except wrapper`, možemo pokupiti i greške:
- `try:`
- `with open(file_path, 'mode') as f:`
- `...`
- `except Exception as e:`
- `print(f"Error! {e}")`

3.4.5. najčešće metode za rad s datotekama:

- `close()` - zatvara file
- `flush()` - prazni buffer
- `read(size)` - vraća sadržaj filea, `size`: broj byteova koji treba vratiti, default -1 = cijeli file
- `readable()` - vraća da li se file može čitati, `True` ako može, `False` ako ne može
- `readline(size)` - vraća jedan redak iz filea, `size`: broj byteova koji treba vratiti, default -1 = cijeli redak
- `readlines(hint)` - vraća listu redaka iz filea, `hint`: max broj byteova koji vraća, default -1 = svi retci
- `seek(offset, whence)` - mijenja poziciju, `offset` od ref. točke `whence`: 0 - početak, 1 - trenutno, 2 - kraj
- `seekable()` - vraća da li je dopušten `seek`, `True` ako može, `False` ako ne može
- `tell()` - vraća poziciju kursora, broj byteova od početka filea
- `writable()` - vraća da li se u file može zapisivati, `True` ako može, `False` ako ne može
- `write(byte)` - zapisuje string u file, pozicija ovisi je li otvoren kao "w" (briše sve) ili "a" (dodaje na kraj)
- `writelines()` - zapisuje listu stringova u file, pozicija ovisi kako je otvoren (isto kao `write`)

4. JSON - JavaScript Object Notation

4.1.1. *tekstualni tip datoteke, čitljiv ljudima i računalima, format zapisa sličan kao dictionary*

4.1.2. *standardni format razmjene podataka na internetu, zamijenio kompliciraniji XML*

- `{ "key": "value",`
- `"key": ["value1", "value2",...],`
- `"key": {"key": "value1", "key": "value2",...},`
- `...}`

4.1.3. *za rad treba uključiti built-in modul:*

- `import json`

4.1.4. *učitavanje JSON sadržaja:*

- `var = json.load(fileReader)` - deserialize `.read()` supporting file-like object
- `var = json.loads(neki_string)` - deserialize `str/byte/bytearray`

4.1.5. *kreiranje tekst varijable s JSON sadržajem:*

- `var = json.dump(data_dict, fileWriter)` - serialize stream to a `.write()` supporting file-like object
- `var = json.dumps(data_dict)` - serialize stream to a JSON-like string

4.1.6. *učitavanje iz datoteke:*

- `try:`
- `with open(file_path, 'r') as fileReader: var = json.load(fileReader)`
- `except: ...`

4.1.7. *zapisivanje u datoteku:*

- `try:`
- `with open(file_path, 'w') as fileWriter: json.dump(var, fileWriter)`
- `except: ...`

5. Podaci na Internetu

5.1. *Web Scraping - "struganje" ili "grebanje" podataka s Interneta*

5.1.1. *Što je dopušteno? Definirano unutar "robots.txt" koja se nalazi u početnoj mapi svake stranice*

5.2. *Server - client komunikacija*

5.2.1. *client (aplikacija na korisničkoj strani) pomoću URL-a pristupa sadržaju na serveru*

- *client je aplikacija koja šalje zahtjeve (request) za sadržajem*

5.2.2. *server je računalo na kojem se nalazi sadržaj*

- *server vraća odgovore (response) s traženim podacima*

5.2.3. *komunikacija je definirana protokolima (npr. HTTPS = HTTP + TLS/SSL)*

5.2.4. *odgovor je najčešće formatiran kao HTML ili JSON*

5.2.5. *http request - response komponente:*

- *1. HTTP REQUEST*
- *1.1. request header:*
 - *metoda koja označava šalje li se sadržaj na server ili se traži od servera*
 - *metode: GET = dohvat, POST = kreiranje, PUT = nadogradnja, DELETE = brisanje*
 - *URI - adresa servera s kojim se povezujemo, uključivo protokol komunikacije*
 - *opcije koje definiraju jezik dokumenta, način autorizacije, itd.*
- *1.2. request body*
- *sadržaj (opcionalno)*
- *2. HTTP RESPONSE*
- *2.1. response header*
 - *status code (npr. najpoznatiji 404 Not Found)*
 - *dodatne informacije o protokolu, statusu konekcije, serveru, itd.*
- *2.2. response body*
- *sadržaj koji smo tražili od servera*

- 5.3. html - format prikaza web stranica
- 5.3.1. tags/oznake - označavaju tip podatka koji se nalazi između početne i krajnje oznake
 - `<html>...</html>` - početak i kraj stranice
 - `<head>...</head>` - dio koji čuva meta podatke o dokumentu npr. naslov, opis, autor, itd.
 - `<body>...</body>` - dio unutar kojeg se nalazi sadržaj dokumenta
 - `<h1>...</h1>` - heading, `<p>...</p>` - paragraph, `<div>...</div>` - divider, ...
- 5.4. urllib - osnove
- 5.4.1. za rad treba ubaciti built-in modul:
 - `import urllib`
- 5.4.2. konekciju radimo koristeći `urlopen`:
 - `konekcija = urllib.request.urlopen(URL)`
- 5.4.3. dohvaćamo sadržaj koristeći konekciju:
 - `sadržaj = konekcija.read().decode()`
- 5.4.4. dohvaćenim sadržajem manipuliramo kao stringom
- 5.5. urllib - praktična sintaksa
- 5.5.1. za rad je potrebno uključiti modul:
 - `import urllib.request, urllib.parse`
- 5.5.2. otvaranje url-a:
 - `connection = urllib.request.Request(URL)`
 - `try:`
 - `with urllib.request.urlopen(connection) as response:`
 - `site = response.read().decode()`
 - `except: ...`
- 5.5.3. google search:
 - `URL = f'https://google.com/search?q={urllib.parse.quote(postavljeni_upit).encode('utf-8')}`
 - `headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; WOW64;rv:12.0) Gecko/20100101 Firefox/12.0'}`
 - `connection = urllib.request.Request(URL, headers=headers)`
 - `try:`
 - `with urllib.request.urlopen(connection) as response:`
 - `data = response.read().decode()`
 - `except: ..`
- 5.5.4. ako se traži SSL context:
 - `import ssl`
 - `context = ssl._create_unverified_context()`
 - `...with urllib.request.urlopen(connection, context) as response`
- 5.6. requests - osnove
- 5.7. jednostavna sintaksa, koristi direktne metode za slanje DELETE, GET, HEAD, PATCH, POST i PUT zahtjeva
- 5.7.1. `.delete(url,args),get(url,params,args),head(url,args),patch(url,data,args),.post(url,data,json,args),.put(url,data)`
- 5.7.2. `GET: requests.get(url, params={key:value}, allow_redirects=True, auth=None, cert=None, cookies=None, headers=None, proxies=None, stream=False, timeout=None, verify=True)`
- 5.8. requests - praktična sintaksa
- 5.8.1. za rad je potrebno instalirati i uključiti modul:
 - `import requests`
- 5.8.2. otvaranje url-a:
 - `response = requests.get(URL)`
 - `print(response.status_code)`
 - `print(response.content)`
 - `print(response.headers)`
 - `print(response.text)`

- 5.9. Beautiful Soup
- 5.9.1. modul za jednostavnije čitanje (parsing) html i xml dokumenata
- 5.9.2. potrebno ga je instalirati (pip ili conda install BeautifulSoup4)
- 5.9.3. za rad je potrebno ubaciti modul
- `from bs4 import BeautifulSoup`
- 5.9.4. za učitavanje sadržaja možemo koristiti urllib:
- `konekcija = urllib.request.urlopen(URL)`
 - `sadrzaj = konekcija.read().decode()`
- 5.9.5. alternativno, možemo koristiti i requests:
- `sadrzaj = requests.get(URL).content`
- 5.9.6. dohvaćeni sadržaj ubacujemo u parser:
- `podaci = BeautifulSoup(sadrzaj, 'html.parser')`
 - `paragrafi = podaci.find_all('p')`
 - `linkovi = podaci.find_all('a')`
 - `naslov = podaci.find('h1').get_text()`
- 5.9.7. nakon parsiranja elemenata, radimo s njima kao sa stringovima:
- `for paragraf in paragrafi: print(paragraf)`
- 5.9.8. CSS selekcija (primjer: <https://books.toscrape.com/>):
- `selekcija = sadrzaj.select('.CSS_class_name')`
- 5.9.9. PRIMJER - book scraping:
- `import requests, os, sys`
 - `from bs4 import BeautifulSoup`
 -
 - `opis_ocjena = {"One": "***", "Two": "****", "Three": "****", "Four": "*****", "Five": "*****"}`
 - `def get_ocjena(tag):`
 - `for (naziv, broj_zvezdica) in opis_ocjena.items():`
 - `if naziv in tag["class"]:`
 - `return broj_zvezdica`
 -
 - `sadrzaj = BeautifulSoup(requests.get("http://books.toscrape.com").content, "html.parser")`
 - `cijene = sadrzaj.select('.price_color')`
 - `naslovi = sadrzaj.select('.product_pod h3 a')`
 - `ocjene = sadrzaj.select('.star-rating')`
 -
 - `with open(os.path.join(sys.path[0], 'books.csv'), 'w', encoding='utf-8') as fw:`
 - `for (cijena, naslov, ocjena) in zip(cijene, naslovi, ocjene):`
 - `fw.write(f"{naslov['title']};{cijena.string};{get_ocjena(ocjena)}\n")`
- 5.10. WebAPI - podaci u json formatu
- klasične web stranice koriste ljudi, takve stranice isporučuju html
- WebAPI stranicama pristupaju računala, takve stranice isporučuju json

6. Baze podataka

- 6.1. skup međusobno povezanih tablica s podacima (veze među tablicama = relacije)
- 6.2. najčešće korišteni tipovi relacija:
- 6.2.1. *one-to-many*: jedna kategorija ima puno proizvoda, svaki proizvod ima jednu kategoriju
- 6.2.2. *many-to-many*: jedan djelatnik može nazvati puno korisnika, jedan korisnik može dobiti pozive više djelatnika
- 6.2.3. *one-to-one*: jedan djelatnik ima samo jednu ID karticu, jedna ID kartica dodjeljuje se jednom djelatniku
- 6.3. SQL
- 6.3.1. *Structured Query Language*, upitni programski jezik visoke razine, dizajniran za rad s bazama podataka.
- 6.3.2. *Query (Upit)* - naziv za naredbe za manipulaciju podacima
- 6.3.3. *CRUD* - *create, retrieve/read, update, delete*
- *INSERT INTO [imeTablice] (kolona1,kolona2) VALUES (vrijednostKolona1,vrijednostKolona2)*
 - *SELECT [kolona1],[kolona2] FROM [imeTablice];*
 - *UPDATE [imeTablice] SET kolona1=novaVrijednostKolona1 WHERE kolona2=vrijednostKolona2*
 - *DELETE FROM [imeTablice] WHERE kolona2=vrijednostKolona2*
- 6.3.4. Dva najzastupljenija tipa baza podataka:
- *Relacijske baze* - podaci su podijeljeni u tabele i međusobno su povezani relacijama u obliku nekih identifikatora. Podaci su konzistentni, ne ponavljaju se, pohrana podataka je otporna na ispade sustava.
 - Najčešće korištene SQL DB: IBM DB2, MySQL, MSSQL Server, ORACLE, PostgreSQL, SQLite
 - *NoSQL baze* - baze primarno namijenjene za visoke performanse, pohranjuju podatke kao cjelinu, bez da ih podijele po tabelama i povežu relacijama, podaci su pohranjeni u drugačije oblike od tabela
 - Najčešća organizacija baze je uz korištenje *collections* (slično tablicama), *documents* (setovi slični JSON-u), *graphs* (organizacija podataka u stabla)
 - Najpoznatije NoSQL DB: MongoDB, Cassandra, Redis, Couchbase, ...
- 6.3.5. Organizacija podataka u bazi:
- *Table* = tablica koja čuva podatke o objektima
 - *Columns* = stupci, slične svojstvima u klasi
 - *Rows* = redovi, predstavljaju objekte (*records*)
 - *Recordset* = kolekcija više redova (dio tablice)
- 6.3.6. SQL osnovne naredbe:
- 6.3.7. kreiranje tablice: *CREATE TABLE nazivTablice (nazivKolone TIP, ...)*
- npr. *CREATE TABLE Djelatnici (id INT PRIMARY KEY, ime VARCHAR(50), dob int, ulica VARCHAR(150));*
- 6.3.8. brisanje tablice: *DROP TABLE IF EXISTS nazivTablice*
- npr. *DROP TABLE IF EXISTS Djelatnici*
- 6.3.9. zapisivanje vrijednosti: *INSERT INTO nazivTablice (nazivKolone1, nazivKolone2 ...) VALUES (val1, val2, ...);*
- npr. *INSERT INTO Djelatnici (djelatnikId, ime, prezime) VALUES (1, "Petar", "Perić");*
- 6.3.10. dohvat podataka: *SELECT nazivKolone1, nazivKolone2, ... FROM nazivTablice WHERE uvjet;*
- npr. *SELECT ime, prezime FROM Djelatnici WHERE ime LIKE "Pet%";*
- 6.3.11. ažuriranje vrijednosti: *UPDATE nazivTablice SET nazivKolone1=val1, nazivKolone2=val2,... WHERE uvjet;*
- npr. *UPDATE Djelatnici SET ime = "Petar Krešimir", prezime = "Perić" WHERE djelatnikId = 1;*
- 6.3.12. brisanje vrijednosti: *DELETE FROM nazivTabele WHERE uvjet;*
- npr. *DELETE FROM Djelatnici WHERE djelatnikId = 1;*

- 6.4. SQLite
- 6.4.1. mala, jednostavna, brza, SQL Database aplikacija
- 6.4.2. ne traži instalaciju - uključen u Python instalaciju i većinu Linux distribucija, uključujući i MacOS

6.4.3. PRIMJER: baza podataka zaposlenika

- `import sqlite3, os, sys`
- `query_create = "CREATE TABLE IF NOT EXISTS Employees (`
- `id INTEGER PRIMARY KEY, name TEXT NOT NULL, EMAIL TEXT NOT NULL UNIQUE);"`
- `query_insert = "INSERT INTO Employees (name,email) VALUES (?,?)"`
- `values = [('Mate Matic', 'mate.matic@mail.com'), ('Ana Anić', 'aanic@hotmail.com'),...]`
- `query_select = "SELECT * FROM Employees WHERE id=?"`
- `values = (2,)`
- `query_update = "UPDATE Employees SET name=?,email=? WHERE id=?"`
- `values = ('Ana Anic Matic', 'aanic@hotmail.com',2)`
- `query_delete = "DELETE FROM Employees WHERE id=?"`
- `query_drop = "DROP TABLE IF EXISTS Employees"`
- `file_path = os.path.join(sys.path[0], "db_name.db")`
- `try:`
- `conn = sqlite3.connection(file_path)`
- `cursor = conn.cursor()`
- `cursor.execute(query,values) # ako postoje vrijednosti izdvojeno od queryja`
- `records = cursor.fetchall() # ako se nešto vadi iz baze`
- `conn.commit() # ako se nešto mijenja u bazi`
- `cursor.close()`
- `except sqlite3.Error as e:`
- `print(f"Error! {e}")`
- `finally:`
- `if conn:`
- `conn.close()`

7. GUI

- prva polovica 1970-ih: Xerox PARC, veliki napredak Apple Lisa Macintosh, 90-ih postaje standard
- najrašireniji je MS Windows, strelovita ekspanzija je krenula s Win 95

7.1. tkinter

7.1.1. Tk Interface - GUI modul uključen u osnovnu instalaciju Pythona

7.1.2. IDLE aplikacija je razvijena pomoću tkinter modula

7.1.3. za rad je potrebno ubaciti built-in modul:

- `import tkinter as tk`

7.1.4. osnovne provjere: `tkinter.TkVersion`, `tkinter._test()`

7.1.5. osnovni objekt (root window) je `Tk()`:

- `rootWindow = tk.Tk()`
- svaki slijedeći element mora imati naveden element koji mu je roditelj
- `Tk()` je roditelj za `root_window` jer se `root_window` nalazi unutar `Tk()`

7.1.6. elementi se smještaju u hijerarhijski odnos koristeći `widgete`:

- svaki `widget` je Python klasa sa specifičnim svojstvima i metodama, neke su zajedničke
- `window` = glavni prozor aplikacije

- *frame* = okvir za grupiranje widgeta
- *label* = prikaz teksta i slika
- *button* = pokretanje aktivnosti
- *checkboxbutton* = da/ne izbor
- *radiobutton* = izbor vrijednosti
- *entry* = polje za unos teksta samo u jednoj liniji
- *listbox* = okvir za prikaz liste (selektabilnih) podataka
- *text* = unos dužeg teksta s više linija

7.1.7. *geometry manager: raspored elemenata unutar prozora*

- *pack*: slaže widgete u okviru jedan ispod ili pored drugog
- *grid*: sličan tablici, koristi kolone i redove, najčešće korišten
- *place*: koriste koordinate prozora u pixelima, slabo korišten

7.1.8. *smještaj komandnog botuna na formi:*

- `tk.Button(rootWindow,text="Hello World").pack()`

7.1.9. *prozor se pokreće kroz main loop:*

- `rootWindow.mainloop()`

7.1.10. *event handling*

- *event* = događaj, npr. klik na gumb, početak unosa teksta, hover mišem i slično
- *event handling* = dio koda koji se izvršava kada se dogodi Event

7.1.11. *PRIMJER: osnovni prozor (poruka, unos vrijednosti, checkbox + oznaka, gumb)*

- `import tkinter as tk`
- `from tkinter import ttk, messagebox`
-
- `class frm_default(ttk.LabelFrame):`
- `def __init__(self,master):`
- `self.root = master`
- `super().__init__(master)`
- `self.configure_basic()`
-
- `self.unos_var = tk.StringVar() # tekst varijable ...`
- `self.potvrda = tk.IntVar() # brojčane varijable ...`
-
- `self.attach_widgets()`
-
- `def configure_basic(self):`
- `self.configure(style='frm_okviri.TLabelframe',text='frame title')`
- `self.unos_var.set("")`
- `self.potvrda.set(0)`
-
- `def set_value(self, value): self.unos_var.set(value)`
- `def get_value(self): return self.unos_var.get()`
- `def action(self): messagebox.showinfo("Status", f"unos: {self.unos_var}, potvrda: {self.potvrda}")`
-
- `def attach_widgets(self):`
- `lbl_poruka = ttk.Label(self,text='tekst',style='lbl_poruke.TLabel')`
- `lbl_poruka.place(x=000,y=000,height=00,width=00,bordermode='ignore')`
-
- `lbl_oznaka = ttk.Label(self,text='tekst',style='lbl_oznake.TLabel')`
- `lbl_oznaka.place(x=000, y=000,height=00,width=00,bordermode='ignore')`
-
- `chk_potvrda = ttk.Checkbutton(self,style='potvrde.TCheckbutton',variable=self.potvrda)`
- `chk_potvrda.place(x=000,y=000,height=00,width=00,bordermode='ignore')`
-

```

ent_unos = ttk.Entry(self, style='ent_unosi.TEntry', textvariable=self.unos_var)
ent_unos.place(x=000, y=000, height=00, width=00, bordermode='ignore')

btn_gumb = ttk.Button(self)
btn_gumb.place(x=000, y=000, height=00, width=00, bordermode='ignore')
btn_gumb.configure(text='tekst', style='btn_gumbi.TButton', command=self.action)

class tkRoot(tk.Tk):
    def __init__(self):
        super().__init__()
        self.configure_basic()
        self.style = self.style_config()
        self.attach_default_frame()

    def configure_basic(self):
        self.title("TITLE")
        self.resizable(0, 0)

    def style_config(self):
        self.style = ttk.Style(self)
        self.style.configure('.', font='TkDefaultFont')
        self.style.configure('frm_okviri.TLabelframe', relief='flat')
        self.style.configure('btn_gumbi.TButton', relief='groove', compound='center', font=('Segoe UI', 12))
        self.style.configure('lbl_poruke.TLabel', background='white', relief='solid', font=('Segoe UI', 9),
                               padding(2, 2, 2, 2), anchor='nw', justify='left', compound='left')
        self.style.configure('lbl_oznake.TLabel', relief='flat', font=('Segoe UI', 9),
                               anchor='w', justify='left', compound='left')
        self.style.configure('chk_potvrde.TCheckbutton', compound='left')
        self.style.configure('ent_unosi.TEntry', font=('Segoe UI', 9), compound='left')

    def attach_default_frame(self):
        self.geometry("000x000")
        self.frm_default = frm_default(self)
        self.frm_default.place(x=0, y=0, height=000, width=000)

    def tksleep(self, t): # emulates time.sleep(seconds)
        var = tk.IntVar(self)
        self.after(int(t*1000), var.set, 1)
        self.wait_variable(var)

class App():
    def __init__(self):
        try: self.interface_root = tkRoot()
        except Exception as e: print(f"Error! {e}")

    def run(self):
        self.interface_root.mainloop()

def main():
    try: App().run()
    except Exception as e: print(f"Error! {e}")

if __name__ == '__main__': main()

```

PYTHON U PODRUČJU INTERNET STVARI

- *IoT - Internet of Things: naziv za mrežu elektroničkih sustava povezanih putem interneta*
- *sustavi se sastoje od hardware-a i software-a*

1. IoT povijest

- 1.1. 1973. Mario Cardullo patentirao prvi RFID
- 1.2. 1982. Carnegie Mellon University prvi samoposlužni automat za piće povezan na internet (Coca-Cola)
 - *mogao je javiti kada nedostaje pića i prepoznati jesu li dodani napici hladni ili ne*
- 1.3. 1989. toster povezan na internet predstavljen na konferenciji Interop '89
- 1.4. 1991. prva kamera povezana na internet (snimala je aparat za kavu u CERN-u)
- 1.5. 1998. osnovan Bluetooth Special Interest Group
 - *2 godine prije udružili se Intel, Nokia i Ericsson oko standardiziranja protokola, bluetooth je bilo kodno ime*
- 1.6. 2000. prvi hladnjak povezan na internet (LG)

2. Ohmov zakon ($U = I \times R$)

- 2.1. odnos napona (U [V - volt]), jakosti struje (I [A - amper]) i električnog otpora (R [Ω - ohm])
- 2.2. većina računalnih komponenti radi na 5V, neke (najčešće senzori) rade na 3.5V

3. IoT računala

- 3.1. više različitih vrsta: Raspberry Pi, Arduino, Tessel, NVidia Jetson, SimpleLink, BeagleBone, ...
- 3.2. Komponente u IoT-u:
 - 3.2.1. *Microcontroller (npr. Arduino) ili Single-board computer (npr. Raspberry Pi)*
 - *na microcontroller su spojeni senzori (žicom) ili je napravljen custom PCB sa zalemljenim sensorima*
 - *najčešće upogonjen baterijom, potrebno paziti kako napraviti implementaciju radi štednje energije*
 - *takvih uređaja može biti mnogo u sustavu, a spajanje svakog na internet može biti skupo ili nemoguće - koristi se radio modul specifičnih frekvencija gdje putem radio signala svi uređaji (node-ovi) šalju digitalne podatke pretvorene iz fizikalnih veličina (senzori) do centralnog mjesta (collector/gateway) - najčešće Raspberry Pi koji ima konekciju na internet - da distribuira podatke u cloud na obradu*
 - 3.2.2. *Radio veza (radio modul - npr. LoRa, Sigfox, XBee)*
 - 3.2.3. *Internet (cloud - sa softwareom)*

4. Elektronički senzori

- 4.1. komponente sa sposobnošću pretvaranja fizikalne pojave iz okoline u električni signal
- 4.2. ovisno o intenzitetu vanjskog utjecaja mijenjaju se svojstva materijala senzora što uzrokuje promjenu vrijednosti električnog otpora ili kapaciteta senzora, a što omogućuje mjerenje vrijednosti pojave
- 4.3. najčešće korišteni senzori: vlažnost, temperatura, barometar, detektor plina, detektor pokreta, ultrazvučni mjerač udaljenosti, GPS receiver, žiroskop, wireless, IC dioda (daljinski upravljač), motor, kamera, ...
- 4.4. Raspberry Pi
 - 4.4.1. *koristi se kao računo koje prikuplja podatke s drugih, još manjih, računala, obrađuje ih te ih ovisno o namjeni prikazuje na ekran ili šalje u Cloud*
 - 4.4.2. *Priključci: USB-C, 2,3, Gbit Ethernet (UTP), Wi-Fi, micro HDMI (do 2x4K monitora), GPIO pinovi, microSD*
 - *GPIO = General Purpose Input / Output, dio preko kojeg se odvija kom. prema sensorima*
 - 4.4.3. *OS: Raspbian (za ARM procesore), na bazi Linux Debian, nema instalacije nego se radi imaging SD kartice*
 - *linux terminal: pwd, ls, mkdir, rmdir, rm, cp, mv, touch, ... ping, hostname, ifconfig*
 - *ugrađena podrška za Python, Java, Scratch programske jezike*
 - *Python IDE: Thonny, Geany, može se instalirati i drugi, npr. VSCode*
 - *emulator senzora na virtualnoj instalaciji Raspbiana je Sense HAT Emulator*
 - 4.4.4. *Zero W: manjih dimenzija, bez USB priključaka, ali s ugrađenim Wi-Fi LAN i Bluetooth modulima*
 - 4.4.5. *Pico: najmanje, namijenjeno povezivanju sa sensorima i slanju očitavanja na veće Raspberry Pi računalo*

- 4.5. Arduino
- 4.5.1. Uno: najčešća verzija, nema OS nego predefinirane instrukcije za pokretanje u EEPROM memoriji, dio memorije dodijeljen korisniku za pohranu programa (pisan najčešće u C++ jeziku)
- 4.5.2. open source pločica - podloga za izradu vlastite verzije Arduino pločice
- 4.5.3. Micro: mala verzija, veličine 48x18 mm
- 4.5.4. Nano: jedna od najmanjih, nešto manja od Micro, ima integriran mikrofon, senzore temperature, vlažnosti i tlaka, Bluetooth modul, Proximity i Gesture senzor - senzore pokreta, vibracije i orijentacije
- 4.5.5. CROduino Pico: hrvatska inačica Arduino pločica
- 4.6. NVidia Jetson
- 4.6.1. Nano: snažan procesor, pogodno za AI projekte, većih dimenzija, malo skuplje, impresivnih karakteristika

5. Komunikacija unutar IoT sustava

- 5.1. IoT Sensor = sklop senzor + mini (najčešće Arduino) IoT računalo (osnovni elektronički senzori mogu se spojiti isključivo izravno na neku "pametniju" pločicu, kao što je Arduino)
- 5.2. Tipovi komunikacije
- 5.2.1. Bežična: NFC (Near-Field Comm), RFID (radio valovi), Bluetooth, Wi-Fi, mobilne mreže (3G, LTE, 5G), satelit
- 5.2.2. Žična: Ethernet (UTP kabel), USB, COM (zastarjeli način)
- 5.3. Glavni uvjeti za komunikaciju: potrošnja energije (PE), domet (D), brzina protoka podataka (BPP)
 - npr. RFID, NFC: niska PE, kratki D, mala BPP
 - npr. mobilne mreže, satelit: visoka PE, veliki D, velika BPP

6. Primjena IoT sustava

- 6.1. potrošači - osobna / kućanstva: male meteo stanice, pametni audio sustavi, narukvice, ...
- 6.2. poslovanje - automatizacija u gotovo svim granama industrije, robotizacija, vojska, zdravstvo, transport i logistika, poljoprivreda, okoliš i ekologija, energija, trgovina, oglašavanje, financije, smart city, ...

7. Sense Hat

- 7.1. Lista senzora:
 - Gyroscope / angle velocity sensor
 - Accelerometer / linear accelerometer
 - Magnetometer / magnetic compass
 - Barometer / atmospheric pressure sensor
 - Temperature
 - Relative humidity
 - 8x8 LED matrix display
 - 5 button joystick
- 7.2. Sense HAT Emulator
 - virtualizirani senzori temperature, vlažnosti, tlaka te žiroskop, accelerometer, joystick i 8x8 LED matrica
 - sastavni dio Raspbian OS, za rad u drugim OS treba instalirati modul
 - primjer korištenja: čitanje podataka sa senzora te ispisivanje u konzoli ili na matrici
- 7.2.1. za rad je potrebno uključiti emulator senzora (Sense Hat):
 - import sense_emu # ili:
 - from sense_emu import SenseHat
- 7.2.2. 8x8 matrica LED dioda
 - prikaz podataka na "ekranu" rezolucije 64 px
 - moguć prikaz teksta kao na traci (scroll), u različitim bojama
 - svaka LED dioda može prikazati boju u RGB spektru
 - mogućnost crtanja znakova

7.2.3. NAREDBE

- `class sense_emu.SenseHat()`
- `clear(*args)` - clears matrix with a single colour, default is black, accepts (r,g,b) as args
- `flip_h(redraw=True)` - flips matrix horizontal
- `flip_v(redraw=True)` - flips matrix vertical
- `get_pixel(x,y)` - returns [r,g,b] representing the pixel specified by (x,y), [TL=(0,0), BR=(7,7)]
- `get_pixels()` - returns list containing 64 [r,g,b] lists representing pixels
- `load_image(file_path,redraw=True)` - updates matrix with 8x8 image
- `set_pixel(x,y,*args)` - updates (x,y) pixel with [r,g,b], [TL=(0,0), BR=(7,7)]
- `set_pixels(pixel_list)` - updates pixels with a list containing 64 [r,g,b] representing pixels
- `show_letter(s,text_colour=[255,255,255],back_colour=[0,0,0])` - displays char s on matrix
- `show_message(text_string,scroll_speed=0.1,text_colour=[255,255,255],back_colour=[255,255,255])`
- `get_accelerometer()` - gets orientation in degrees from accelerometer only
- `get_accelerometer_raw()` - x y z raw data in Gs
- `get_compass()` - gets North direction in degrees from magnetometer
- `get_compass_raw()` - x y z raw data in μT (micro teslas)
- `get_gyroscope()` - gets orientation in degrees from gyroscope only
- `get_gyroscope_raw()` - x y z raw data in radians per second
- `get_orientation_degrees(), get_orientation_radians()` - returns dict with (pitch,roll,yaw)
- `get_pressure()` - returns pressure in Millibars
- `get_temperature()` - returns temperature in Celsius
- `get_humidity()` - returns percentage of relative humidity
- `stick` - SenseStick object representing the Sense HAT's joystick

- `class sense_emu.SenseStick()`
- `get_events()` - returns list of joystick events since last call to `get_events()` in order they occurred
- `wait_for_event(emptybuffer=False)` - waits until joystick event becomes available
- `direction_any, direction_middle, direction_down, direction_left, direction_up, direction_right` - f. call on j. use

- `class sense_emu.InputEvent()`
- `namedtuple()` derivative representing a joystick_event
- `timestamp` - time at which it occurred, represented as number of seconds since epoch (like time())
- `direction` - direction of push/release [D= DIRECTION]: D_DOWN,D_LEFT, D_UP,D_RIGHT, D_MIDDLE
- `direction` - alternative constants: 'down', 'left', 'up', 'right', 'middle'
- `action` - push/hold/release [A=ACTION]: A_PRESSED, A_RELEASED, A_HELD
- `action` - alternative constants: 'pressed', 'released', 'held'

7.2.4. PRIMJER - korištenje joysticka:

- `hat=SenseHat()`
- `x=y=4`
- `def update_screen(colour=[255,255,255]):`
- `hat.clear()`
- `hat.set_pixels(x,y,colour)`
- `def clamp(value,min_value=0,max_value=7):`
- `return min(max_value,max(min_value,value))`
- `def move_dot(event):`
- `global x,y`
- `if event.action in ('pressed','held'):`
- `x = clamp(x+{'left':-1,'right':1}.get(event.direction,0))`
- `y = clamp(y+{'up':-1,'down':1}.get(event.direction,0))`
- `update_screen()`
- `while True:`
- `for event in hat.stick.get_events():`
- `move_dot(event)`
- `update_screen()`

PYTHON U PODATKOVNOJ ZNANOSTI

1. Podatkovna znanost je:

- 1.1. interdisciplinarno područje koje koristi znanstvene metode, procese, algoritme i sustave za izlučivanje znanja i spoznaja iz (ne)strukturiranih podataka, primjenu znanja s podataka sa širokog raspona domena aplikacije
- 1.2. povezana je s rudarenjem podataka, strojnim učenjem i velikom količinom podataka
- 1.3. obuhvaća matematičke vještine (linearna algebra, statistika, vjerojatnost, ...), "hakerske" vještine (računalno razmišljanje, poznavanje računalne znanosti te stručnost u području za koje se analiziraju podaci
- 1.4. uči nas radu s podacima, točnije kako:
 - 1.4.1. *prilagoditi format podatka*
 - 1.4.2. *očistiti podatke*
 - 1.4.3. *uzorkovati podatke prema odgovarajućoj skupini*
 - 1.4.4. *komunicirati rezultate kroz vizualizaciju, opis i sažetu interpretaciju rezultata*

2. Podatkovna znanost nije:

- 2.1. Big Data - tehnika za prikupljanje, pohranu i obradu velike količine podataka
- 2.2. Data Mining - tehnika za otkrivanje važnih informacija unutar podataka
- 2.3. Business Intelligence - bavi se analizom poslovnih podataka
- 2.4. AI - tehnika primjene modela dobivenih obradom podataka (forecasting, analiza trendova, ...)

3. Data Science Pipeline:

- 3.1. proces obrade podataka najjednostavnije se prikazuje pomoću lijevka u koji ulaze sirovi podaci, a izlaze pojednostavljene i razumljive informacije:
 - 3.1.1. *prikupljanje podataka*
 - *najčešće automatiziran proces, korištenjem senzora*
 - *nudjenje besplatne usluge koja analizira podatke koje korisnici kreiraju koristeći uslugu*
 - *razne ankete i upitnici*
 - 3.1.2. *"čišćenje" i prilagodba podataka*
 - *uočavanje i otklanjanje grešaka u podacima*
 - *ispravljanje krivih formata podataka*
 - *uklanjanje praznina, odnosno nepostojećih zapisa*
 - 3.1.3. *analiza, vizualizacija i interpretacija podataka*
 - *primjenom računalne znanosti (programiranje, aplikacije)*
 - *primjenom matematičkih proračuna (statistika)*
 - *analiziranje podataka kako bismo uočili uzorke i dobili odgovore*
 - 3.1.4. *revizija (povremeno)*
 - *zbog konstantnih promjena povremeno je potrebno napraviti reviziju analize i zaključaka*

4. Python alati:

4.1. Anaconda

4.1.1. skup alata i modula specijaliziranih za primjenu u podatkovnoj znanosti

4.1.2. instalacijom Anaconde dobivamo Python te alate i module za rad

4.2. Jupyter Notebook

4.2.1. Grafički IDE za IPython

4.2.2. Pokretanje:

- preko Anaconda Navigatora
- iz konzole (CMD): `jupyter notebook`

4.3. NumPy

4.3.1. naziv proizašao iz Numerical Python, temeljni paket za znanstvene proračune

4.3.2. lakoća pisanja high level jezika (Python) sa snagom i brzinom low level jezika (C)

4.3.3. osnovni objekt: NumPy ndarray - višedimenzionalni vektor

- često se koriste pravokutne i kvadratne (2D) matrice za rad potrebno je instalirati i uključiti modul:
- `import numpy as np` # konvencija

4.4. Pandas:

4.4.1. naziv proizašao iz Panel Data, alat za analiziranje podataka u Pythonu

4.4.2. podatke sprema u DataFrame, u koji možemo ubaciti podatke iz većine poznatih formata (SQL, Excel, CSV, ...)

4.4.3. razlika NumPy i Pandas:

- Pandas je biblioteka za obradu heterogenih tipova podataka
- NumPy je namijenjen za numeričke proračune i orijentiran na numeričke tipove podataka

4.4.4. za rad potrebno je instalirati i uključiti modul:

- `import pandas as pd` # konvencija
- napomena: obično se numpy i pandas uključe zajedno

4.5. Matplotlib

4.5.1. koristi se za vizualizaciju i izradu grafova visoke kvalitete

4.5.2. jednostavna uporaba, moguća ugradnja u bilo koji Python GUI

4.5.3. za rad potrebno je instalirati i uključiti modul:

- `import matplotlib.pyplot as plt` # konvencija

5. NumPy

5.1. Array creation - there are 6 general mechanisms for creating arrays, 3 most important:

5.1.1. conversion from other Python structures (lists and tuples)

- `a1D = np.array([1,2,3,4,5,6,7,8])`
- `a2D = np.array([[1,2,3,4],[5,6,7,8]])`
- `a3D = np.array([[[1,2],[3,4]],[[5,6],[7,8]])`

5.1.2. intrinsic array creation functions

- you should consider the data object type of array elements, which can be specified explicitly
- numeric types: byte (signed/unsigned), int (signed/unsigned) / float / complex ; bit size: 8,16,32,64,...
- non-numeric: datetime/timedelta, unicode strings, raw data (void)
- 1D
 - `np.arange([start=0],stop,[step=1],dtype=None,*,...)` - evenly spaced values with specified steps
 - `np.linspace(start,stop,num=50,dtype=None,...)` - evenly spaced over specified interval
- 2D
 - `np.eye(N,M=N,k=0,dtype=None...)` - defines a 2D identity matrix NxM, k = diag. shift indeks
 - `np.diag(v,k=0)` - extract diag. (v=2D) or construct diag. (v=1D) array, k = diag. in question
 - `np.vander(x,N=len(x))` - Vandermonde matrix (columns of powers of input), N = num. of columns
- nD
 - `np.zeros(shape,dtype=float,...)` - array of given shape and type, filled with zeros
 - `np.ones(shape,dtype=float,...)` - array of given shape and type, filled with ones
 - `np.full(shape,fill_value,dtype=None,...)` - array of given shape and type, filled with fill_value
- random input
 - `from numpy.random import default_rng as rng` - random number generator
 - `rng().integers(low,[high=None],size=None,dtype=np.int64,...)` - random integers low - high or 0 - low
 - `rng().random(size=None,dtype=np.float64,...)` - random [0,1), size = output shape, default is scalar
 - `rng().choice(A,size=None,replace=True,p=0,axis=0,shuffle=True)` - random sample from A, p=prob.
 - `rng().shuffle(x,axis=0)` - modify array or sequence in place by shuffling its contents
 - `rng().permutation(x,axis=0)` - copy and randomly permute x, or return a permuted range
 - `rng().permuted(x,axis=None)` - randomly permute x along axis, each slice shuffled independently
 - `rng().some_distribution(...)` - draws from specified distribution (normal, geometric, poisson, ...)
 - `rng().binomial(n,p,size=None)` - sample from binomial distribution, n trials, p prob. of success
 - `rng().geometric(p,size=None)` - sample from geometric distribution, p - prob. of success
 - `rng().standard_normal(size=None,dtype=np.float64,...)` - sample from $\varphi < np.random.randn(d1,...,dn)$
 - `rng().poisson(lam=1.0,size=None)` - sample from poisson distribution, lam - expected events in interval

5.1.3. replicating, joining, or mutating existing arrays

- create array from existing list:
 - `L = [1,2,3,4]`
 - `A = np.array(L)`
- stacking and block composing
 - `np.repeat(a, repeats, axis=None)` - repeat each element of an array after themselves
 - `np.concatenate((a1,a2,...),axis=0,...)` - join a sequence of arrays along an existing axis
 - `np.hstack(L1,L2,...)` - stacks arrays in sequence horizontally (column wise)
 - `np.vstack(L1,L2,...)` - stacks arrays in sequence vertically (row wise)
 - `np.dstack(L1,L2,...)` - stacks arrays in sequence along third axis (depth wise)
 - `np.column_stack(L1,L2,...)` - take 1D arrays, stack as columns to make single 2D array
 - `np.block(arrays)` - assemble ndarray from nested lists of blocks, dim from last to first
- unstacking and splitting
 - `np.unstack(x,*,axis=0)` - split ndarray into sequence of arrays along given axis
 - `np.split(A,indices_or_sections,axis=0)` - split into sub-arrays along specified axis
 - `np.hsplit(A,indices_or_sections)` - split into sub-arrays horizontally (column wise)
 - `np.vsplit(A,indices_or_sections)` - split into sub-arrays vertically (row wise)
 - `np.dsplit(A,indices_or_sections)` - split into sub-arrays along third axis (depth wise)

5.2. Array attributes

- *A.ndim* - number of dimensions
- *A.shape* - tuple of non-negative integers, number of elements along each dimension
- *A.size* - total number of elements, product of shape items
- *A.dtype* - data type for homogenous arrays
- *A.itemsize* - size of each element of array in bytes
- *A.nbytes* = *A.itemsize* * *A.size* - total bytes consumed

5.3. Array indexing and assignment (<https://numpy.org/doc/stable/user/basics.indexing.html>)

5.3.1. basic indexing

- *single element*: *A[x]* - returns x-th element of A, like Python list
- *number of x dim. in relation to A dim. gives sub-array of appropriate dimensionality*
- * 1D A, scalar x gives element of A at index X: *A* = [1,2,3,4], *x* = 0, *A[x]* = 1
- * 2D A, scalar x gives 1D sub-array at index X: *A* = [[1,2],[3,4]], *x* = 0, *A[x]* = [1,2]
- * 2D A, pair x gives element of A at indices x1,x2: *A*[[1,2],[3,4]], *x* = (0,0), *A[x]* = 1
- *can split pair into double bracket - same action, less efficient*: *A*[[1,2],[3,4]], *A*[0][0] = 1
- *slicing and striding*
- *single slice*: *A[i:j:k]* - selects [i,i+k,+2k,...(m-i)*k where m=q+(r!=0), j-i = qk+r, i+(m-1)k<j
- *x* = np.array([1,2,3,4,5,6,7,8,9]), *x*[1:7:2] = array([1,3,5])
- *negative i and j are interpreted as n+1 and n+j where n is the number of elements*
- *x* = np.array([1,2,3,4,5,6,7,8,9]), *x*[-2:10] = array([8,9])
- *negative k makes stepping go towards smaller indices*
- *x* = np.array([1,2,3,4,5,6,7,8,9]), *x*[-3:3:-1] = array([7,6,5,4])
- *if i is not given it defaults to 0, if j is not given it defaults to n-1, k defaults to 1*
- *x* = np.array([5,6,7,8,9]), *x*[5:] = array([5,6,7,8,9])
- *slicing may be used to set values in array, but, unlike lists, you can never grow the array*
- *dimensional indexing*
- *Ellipsis expands to the number of ":" objects needed for the selection tuple to index all dimensions*
- *A* = array([[[[1],[2],[3]],[[4],[5],[6]]]]), *A* [...,0] = *A*[:, :, 0] = array([[1,2,3],[4,5,6]])
- *each newaxis object expands resulting selection dims by one unit-length dim, alias for None*
- *A[:,np.newaxis,:].shape* = *A[:,None,:].shape* = (2,1,3,1)

5.3.2. advanced indexing

- *integer array indexing*: *A*[np.array([x1,x2,...])] selects items based on their N-dimensional index
- *A* = [10,9,8,7,6,5,4,3,2], *A*[np.array([3,3,1,8])] = [8,8,9,2]
- *A* = [[0,...6],[7,...,13],[14,...,20],[21,...,27],[28,...,34]]
- *A*[np.array([0,2,4],np.array([0,1,2])] = [0,15,30]
- *A*[np.array([0,2,4],1)] = [1,15,29]
- *A*[np.array([0,2,4]) = [[0,...6],[14,...,20],[28,...,34]]
- *boolean array indexing*: *A*[obj] where obj is an array of boolean type (returned from comparisons,etc.)
- *A* = [1,-2,-3,4], *A*[*A*<0] = [-2,-3]
- *A* = [0,1,0,2], *A*[*A*.nonzero()] = [1,2]
- * np.argwhere(*A*) - indices of array elements that are non-zero, grouped by element
- *A* = [[1,...,4],[5,...,8],[9,...,12]], *A*[*A*%2==0] = [2,4,6,8,10,12]
- *A* = [[0,1],[1,1],[1,2]], *A*[*A*.sum(-1)<=2,:] = [[0,1],[1,1]] - rows with sums <= 2
- *A* = np.arange(30).reshape(2,3,5), *A*[np.array([T,T,F],[F,T,T])] = [[0,...,4],[5,...,9],[20,...,24],[25,...,29]]
- *stacking conditions using logical operators (& = and, | = or)*:
- *A* = [0,...,12], *A*[(*A*>2)&(*A*<11)] = [3,4,5,6,7,8,9,10]
- *conditional indexing*
- np.where(condition,[x,y]) - where true yield x, otherwise yield y, x and y - manipulate array values
- np.select(condlist, choicelist, default=0) - condlist - list of conditions, choicelist - list of arrays from which the output elements are taken, same length as condlist, default scalar when all conditions return False

5.4. saving and loading

- np.savetxt(filename,A,format,delimiter=' ',newline='\n',header="",footer="",...) - save array to txt file
- np.loadtxt(filename,converters=None,delimiter=None,skiprows=0,usecols=None,...) - load array from txt file

5.5. Manipulating arrays

5.5.1. manipulating dimensionality

- *newaxis*: increasing dimensions of array by one dimension when used once
- $A = [1, 2, 3, 4, 5, 6]$, $A.shape = (6,)$
- $A1 = A[np.newaxis, :] = [[1, 2, 3, 4, 5, 6]]$, $A1.shape = (1, 6)$
- $A2 = A[:, np.newaxis] = [[1], [2], [3], [4], [5], [6]]$, $A2.shape = (6, 1)$
- *expand_dims*: expand array by inserting a new axis at a specified position
- $A = [1, 2, 3, 4, 5, 6]$, $A.shape = (6,)$
- $B1 = np.expand_dims(a, axis=0) = [[1, 2, 3, 4, 5, 6]]$, $B1.shape = (1, 6)$
- $B2 = np.expand_dims(a, axis=1) = [[1], [2], [3], [4], [5], [6]]$, $B2.shape = (6, 1)$

5.5.2. basic array operations

- basic operators (+, -, *, /, %) can be used directly and are applied item by item
- $A = [1, 2, 3, 4]$, $B = [5, 6, 7, 8]$, $A+B = [6, 8, 10, 12]$, $A-B = [-4, -4, -4, -4]$, ...
- $np.sum(A, axis=None, ...)$ - sums the elements in an array, all or by specified axis
- $np.prod(A, axis=None, ...)$ - multiplies the elements in an array, all or by specified axis
- $np.max(A, axis=None, ...)$ - returns max item in an array, all or by specified axis
- $np.argmax(A, axis=None, ...)$ - returns indices of max values along an axis
- $np.min(A, axis=None, ...)$ - returns min item in an array, all or by specified axis
- $np.argmin(A, axis=None, ...)$ - returns indices of min values along an axis
- $np.unique(A, axis=None, return_index/inverse/counts=False, ...)$ - returns sorted unique items
- $np.sort(A, axis=-1, kind=None, ...)$ - sorts array, kind: {'quicksort', 'mergesort', 'heapsort', 'stable'}

5.5.3. transposing and reshaping

- $A.T$, $np.transpose(A, axes=None)$ - returns transposed matrix
- $np.reshape(A, shape=None, copy=None, ...)$ - returns array reshaped to specified shape

5.5.4. reversing

- $np.flip(A, axis=None)$ - flips array along the specified axis
- $A = np.arange(1, 13).reshape(3, 4)$
- $np.flip(A) = [[12, ..., 9], [8, ..., 5], [4, ..., 1]]$
- $np.flip(A, axis=0) = [[9, ..., 12], [5, ..., 8], [1, ..., 4]]$
- $np.flip(A, axis=1) = [[4, ..., 1], [8, ..., 5], [12, ..., 9]]$
- $np.fliplr(A)$ - flips left-right, $np.flipud(A)$ - flips up-down
- $np.rot90(A, k=1, axes=(0, 1))$ - rotates array 90 degrees, k = number of rotations, $axes$ = rotation plane

5.5.5. flattening

- $np.flatten(A)$ - less memory efficient, creates a copy
- $np.ravel(A)$ - more memory efficient, creates a view

5.6. Working with functions (element wise)

5.6.1. trigonometric functions

- \sin , \cos , \tan , \arcsin , \arccos , \arctan

5.6.2. hyperbolic functions

- \sinh , \cosh , \tanh , $\operatorname{arcsinh}$, $\operatorname{arccosh}$, $\operatorname{arctanh}$

5.6.3. $np.hypot(x1, x2)$ - given legs of right triangle calculates hypotenuse (Euclidian norm)

5.6.4. $np.round(A, d)$ - evenly round A to given d number of decimals

- other rounding functions: rint (to nearest int), floor , ceil , trunc

5.6.5. exponents and logarithms

- \exp , $\exp2 (2^{**x})$, \log (natural), $\log2$, $\log10$

5.6.6. rational routines

- $np.lcm(x1, x2, ...)$ - lowest common denominator
- $np.gcd(x1, x2, ...)$ - greatest common divisor

5.6.7. arithmetic operations

- add, subtract, multiply, divide, mod

5.6.8. power operations

- reciprocal, power, sqrt, square

5.6.9. numerical values

- absolute, negative, positive, sign

- 5.6.10. *logic functions*
 - *all* - test if all along axis are True, *any* - test if any along axis are True
 - *isfinite* (not infinity, not NaN), *isinf* (infinity), *isnan* (not a number)
 - *iscomplex* (complex number), *isreal* (real number) - returns bool array
 - *logical operations*: *logical_and*, *logical_or*, *logical_not*, *logical_xor*
 - *comparison*: *is_close* (equal within a tolerance), *array_equal* - same, *array_equiv* - look same
 - *greater*, *greater_equal*, *less*, *less_equal*, *equal*, *not_equal*
- 5.6.11. *custom functions*
 - *np.fromfunction*(function,shape,dtype...) - construct array by executing a function over each coordinate
- 5.7. Universal function basics
- 5.7.1. *np.ufunc* - functions that operate element by element on whole arrays
 - *np.ufunc.reduce*(A,axis=0) - reduce dim by one by applying ufunc
 - *np.add.reduce*, *np.subtract.reduce*, *np.minimum.reduce*, *np.maximum.reduce*, ...
 - *np.ufunc.accumulate*(A,axis=0) - accumulate the result of applying operator to all elements
 - *np.add.accumulate*, *np.subtract.accumulate*, *np.multiply.accumulate*, *np.divide.accumulate*, ...
- 5.8. Statistics
- 5.8.1. *order statistics*
 - *ptp* - range of values (maximum - minimum)
 - *percentile* - compute the q-th percentile
 - *quantile* - compute the q-th quantile
- 5.8.2. *averages and variances*
 - *average* (weighted), *mean* (arithmetic), *median*
 - *std* (standard deviation), *var* (variance)
- 5.8.3. *correlating*
 - *corrcoef* - Pearson product-moment correlation coefficients
 - *correlate* - cross-correlation of two 1-dimensional sequences
 - *cov* - estimate a covariance matrix, given data and weights
- 5.8.4. *histograms*
 - *histogram* - compute the histogram of a dataset
- 5.9. Linear algebra (numpy.linalg module)
- 5.9.1. *matrix and vector products*
 - *dot* (product of 2 vectors), also: *vdot*, *vecdot*, *inner*, *outer*
 - *matmul* - matrix product of 2 arrays, implements @ operator
 - *matrix_power* - raises a square matrix to integer power
 - *cross* - cross product of 3 element vectors
- 5.9.2. *decompositions*
 - *qr* - qr factorisation of a matrix
 - *svd* - singular value decomposition
- 5.9.3. *matrix eigenvalues*
 - *eig* - compute eigenvalues and right eigenvectors of a square array
 - *eigh* - return eigenvalues and eigenvectors of a complex Hermitian or a real symmetric matrix
- 5.9.4. *norms and other numbers*
 - *norm* - matrix or vector norm, also: *matrix_norm*, *vector_norm*
 - *det* - determinant of an array
 - *rank* - matrix rank using SVD method
 - *trace* - sum along diagonals
 - *diagonal* - return specified diagonals
- 5.9.5. *solving equations and inverting matrices*
 - *solve* - solve a linear matrix equation, or system of linear scalar equations
 - *lstsq* - least squares solution to a linear matrix equation
 - *inv* - compute the inverse of a matrix
 - *pinv* - compute the Moore - Penrose pseudo-inverse

6. Pandas

6.1. Basic data structures in pandas

- *Series*: 1D labeled array holding data of any type (integers, strings, Python objects, ...)
- *DataFrame*: 2D structure that holds data like 2D array or a table with rows and columns
- * NumPy arrays have one dtype, while pandas DataFrames have one dtype per column

6.2. Object creation

6.2.1. Creating a series:

- passing a list of values, let pandas create default RangeIndex:
 - `ss = pd.Series([1,2,3,4,5,6,np.nan,8])`
 - `ss = pd.Series([1,2,3,4,5,6],name="Serija_brojeva")` - 6 records, numerical
 - `ss = pd.Series(["A","B","C","D"],name="Serija_slova")` - 4 records, string
 - `ss = pd.Series([[1,2,3],[4,5,6]],name="Serija_lista")` - 2 records, each a 3 member list
- passing a list of values and an index
 - `ss = pd.Series([1,2,3,4],index=['a','b','c','d'])` - 4 records, numerical, letter indices

6.2.2. Creating a DataFrame:

- passing a NumPy array with a datetime index using `date_range()` and labeled columns:
 - `dates = pd.date_range("20240101", periods=6)` - periods = how many dates
 - `df = pd.DataFrame(np.random.randn(6,4),index=dates,columns=list("ABCD"))`
- passing a dictionary of objects where the keys are column labels
 - `df = pd.DataFrame({"A": value, "B": value, "C": value, "D": value, ...})`
 - * values can be numerical, string, np.array, pd.Series, pd.Categorical, pd.Timestamp, ...

6.2.3. Resetting the index

- `ss.reset_index(level=None, *, drop=False,...)` - generate new series with index reset
- `df.reset_index(level=None, *, drop=False,...)` - reset df index or a level of it, use default
- `drop` = whether to try to insert the index as a column: `False` = insert, `True` = remove

6.3. Viewing data

6.3.1. Head and Tail

- `df.head(n=5)` - return the first *n* rows, for negative values returns all rows except last *|n|* rows
- `df.tail(n=3)` - return the last *n* rows, for negative values returns all rows except first *|n|* rows

6.3.2. Summary

- `df.describe(...)` - generate descriptive statistics (central tendency, dispersion, dist. shape, ...)

6.3.3. DataFrame attributes

- `df.index` - displays the index attribute - index labels of the DataFrame
- `df.columns` - displays the columns attribute - columns labels of the DataFrame

6.3.4. Sorting data

- `df.sort_index(*, axis=0, ascending=True, inplace=False, kind="quicksort", na_position="last",...)`
- `axis` 0 = rows, axis 1 = columns
- `kind`: {"quicksort", "mergesort", "heapsort", "stable"}
- `df.sort_values(by=*, axis=0, ascending=True, inplace=False, kind="quicksort", na_position="last",...)`
- `by`: str or list of str, name or list of names to sort by - if given a list, it represents crit. order
- `df.sort_values(by=['crit_1','crit_2',...], ascending=False)` - sort by `crit_1` then by `crit_2`,... descending
- `axis`: 0 > by: index levels and/or column labels. 1 > by: column levels and/or index labels

6.4. Selecting data

6.4.1. Getting data

- *passing a single label selects columns and yields a Series*
- `df["A"], df.A` - return column "A" as Series
- *passing a slice : selects matching rows*
- `df[0:3]` - return first 3 rows as DataFrame
- `df["20230101":"20240101"]` - returns rows within 2023 as DataFrame
- *selection by label using loc*
- `df.loc[dates[0]]` - returns all columns, row which corresponds to first date in date range
- `df.loc[dates[0], "A"]` - returns column "A", row which corresponds to first date in date range
- `df.loc[:, ["A", "B"]]` - returns columns "A" and "B", rows for the whole date range
- `df.loc["20230101":"20240101", ["A", "B"]]` - returns columns "A" and "B", rows in the slice
- * loc function includes both endpoints when slicing
- * faster access to scalar using `at`: `df.at[dates[0], "A"]` - equivalent to `df.loc[dates[0], "A"]`
- *selection by position using iloc*
- `df.iloc[3]` - returns row at index number 3 (starting index is 0), all columns
- `df.iloc[3:5, 0:2]` - returns rows at indices number 3 and 4, columns 0 and 1
- `df.iloc[[1, 2, 4], [0, 2]]` - returns rows at indices number 1, 2, 4, columns 0 and 2
- `df.iloc[1:3, :]` - returns rows at indices 1 and 2, all columns
- `df.iloc[:, 1:3]` - returns all rows, columns 1 and 2
- `df.iloc[1, 1]` - returns value at row index 1, column 1
- * iloc function includes only the first endpoint when slicing
- * faster access to scalar using `iat`: `df.iat[1, 1]` - equivalent to `df.iloc[1, 1]`
- *boolean indexing:*
- `df[df["A"] > 0]` - returns all rows where values in column "A" are greater than 0
- `df[df > 0]` - returns the whole df, cells with values less than 0 converted to NaN
- `df[df["E"].isin(["two", "four"])]` - returns all rows where value in column "E" is "two" or "four"

6.4.2. Setting

- *adding a new column auto aligns the data by the indices*
- `s1 = pd.Series([1, 2, 3, 4, 5, 6], index = pd.date_range("20240101", periods=6))` - append col, align dates
- *by label: `df.at[dates[0], "A"]` - sets the value in first row, column "A" to 0*
- *by position: `df.iat[0, 1] = 0`*
- *by assigning with a NumPy array: `df.loc[:, "D"] = np.array([5]*len(df))` - sets all values in column "D" to 5.0*
- *by boolean indexing: `df[df > 0] = -df` - makes all positive values in df negative with same abs value*
- `df.replace(find, value, *, inplace=False, ...)` - replaces found (num, str, list, dict) by specified value
- `df.drop(labels, *, axis=0, index=None, columns=None, ...)` - return Series with spec. index labels removed

6.5. Missing data

6.5.1. Reindexing

- `df.reindex(labels=None, *, index=None, columns=None, axis=None, method=None, copy=None, level=None, fill_value=None, limit=None, tolerance=None)`
- *index = new labels for the index, columns = new labels for the columns, axis = axis to target (name or number, instead of rows and cols), method = for hole filling, copy = create copy, default True, fill_value = value to use for missing values, default NaN*
- `df1 = df.reindex(index=dates[0:4], columns=list(df.columns)+["E"])`
- *takes first 4 rows from df and all columns + adds new column "E"*

6.5.2. Dealing with NaN

- `df.dropna(*, axis=0, how, thresh, subset=None, inplace=False, ignore_index=False)` - remove missing values
- *how = {"any", "all"} - remove row if any NaN or all NaN or thresh = remove row if at least thresh NaN*
- `df.dropna(how="any")` - removes all rows which contain at least 1 NaN
- `df.fillna(value=None, *, method=None, axis=None, inplace=False, limit=None)` - fill missing values
- *value = how to fill holes, accepts var, dict, Series, DataFrame to separate filling by index and column*
- `df.fillna(value=5)` - replaces all NaN in df with the value 5.0
- `pd.isna(obj)` - creates boolean mask where NaN values are True, obj is the object to check for NaN values
- `pd.isna(df)` - returns boolean mask with all NaN values in df are labeled True

6.6. Operations

- ops in general exclude missing data

6.6.1. Basic functions

- `df.abs()` - return series or df with absolute numeric values of each element
- many math functions available: `min`, `cummin`, `max`, `cummax`, `sum`, `cumsum`, `prod`, `cumprod`, `pow`, ...
- return index (row label) of located value: `idxmin`, `idxmax`
- `df.apply(func,axis=0,...)` - apply a function along an axis of the data frame
- `df.apply(lambda x: x*2+3)` - user defined lambda functions
- `df.transform(func, axis=0, *, **)` - call func on self, producing a DataFrame with same axis shape as self
- `df.transform(lambda x: x+1)` - adds 1 to each value in df
- `s.transform([np.sqrt,np.exp])` - list of input functions which transform the series
- `df.groupby("Date")["Data"].transform('sum')` - sums values on same dates

6.6.2. Statistics

- axis: 0 - per column, default or 1 - per row
- `df.mean(axis=0,skipna=True,numeric_only=False,...)` - calculates mean of rows or columns
- many stats functions available: `corr`, `cov`, `kurt`, `median`, `mode`, `quantile`, `skew`, `std`, `var`, ...
- `df.agg(func,axis=0,*,**)` - aggregate using one or more ops over the specified axis
- `df.agg(lambda x: np.mean(x)*5)` - user defined lambda functions over rows or columns
- `df.agg(['sum','min'])` - common functions applied to all rows or columns stacked into lists
- `df.agg({'A':['sum','min'],'B':['min','max']})` - common functions per column (rep. by dicts)
- `df.agg(x=('A','max'),y=('B','min'),z=('C','mean'))` - different functions and index renaming

6.6.3. Counting

- `df.value_counts()` - returns number of repeats of a value in a series

6.6.4. String Methods

- string processing methods set is accessed through the `str` attribute
- `s.str.lower()` - converts all strings in the series to lowercase

6.6.5. Pivot Tables

- `pd.pivot_table(data, values, index, columns, aggfunc='mean',fill_value=None, dropna=True,margins=False)`
- `data = df`, `values = columns to aggregate`, `index, columns = keys to group by on p.t.` `index, columns`
- `aggfunc = aggregation function, list of functions, dict of functions`
- `margins = add special "All" rows and columns with partial group aggregates across categories`
- `pd.pivot_table(df, values="D", index=["A","B"],columns=["C"], aggfunc="sum")`
- aggregate values in column "D", use sum as the aggregate function
- separate columns by values in column "D", separate rows by values in columns "A" and "B",

6.7. Splitting and Merging (operating with multiple objects)

6.7.1. Concatenating

- `pd.concat(list_of_objs, axis=0, join='outer', ignore_index=False, keys=None, sort=False,...)`
- axis - 0 (rows) or 1 (columns), join - 'outer' or 'inner', ignore_index - resets index for concat. df
- `pd.concat(list_of_df,axis=0 or 1)` - concatenates dfs by rows or columns

6.7.2. Joining

- `pd.merge(left, right, how='inner', on=None, sort=False, copy=None, ...)` - database-style join
- `how = {'left', 'right', 'inner', 'outer', 'cross'}`, `on = names to join on`
- `pd.merge(left, right, on="key")` - merges dfs left and right on column "key"

6.7.3. Grouping

- group by = split into groups based on criteriy + apply function to each group + combine results
- `df.groupby(by=None, axis, as_index=True, sort=True, group_keys=True, dropna=True)`
- `by = determining the groups`, axis = 0 (rows) or 1 (columns), `as_index = group labels as index`
- `df.groupby("A")["C","D"].sum()` - group by column "A" label, apply sum to cols in groups, combine to df
- `df.groupby(...).size()` - number of rows in each group as Series / DataFrame (as_index True / False)

6.7.4. Binary operations

- `df.OPERATION(other, axis='columns',...)` - df OPERATION df, element-wise
- ops: add (addition), sub (subtraction), mul (multiplication), div (division), mod (modulo), pow (power)
- comparing: lt (<), le (<=), eq (==), ne (!=), ge (>=), gt (>) - returns df of bool
- broadcasting is auto applied (scalars to whole df, series to rows, df to cells, dicts to rows or columns)

6.8. Time series

6.8.1. Creating date ranges

- `pd.date_range(start=None, end=None, periods=None, freq=None, tz=None, name=None, unit=None,...)`
- `start` = left bound, `end` = right bound, `periods` = number of periods, `name` = datetimeindex name
- `freq` = offset aliases: `s` = second, `min` = minute, `h` = hour, `W` = week, `MS&ME` = month, `YS&YE` = year

6.9. Manipulating time labels

- *reshaping to fit a different interval:*
- `rng = pd.date_range("1/1/2024", periods=100, freq="s")` - 100 seconds from 2024 new year
- `ts = pd.Series(np.random.randint(0,500,len(rng),index=rng))` - series with 100 random ints 0-500
- `ts.resample("5min").sum()` - sums values to reshape data into 5 minute intervals (300 s)
- *localizing a time series to a time zone:*
- `rng = pd.date_range("3/6/2024 00:00", periods=5, freq="D")` - 5 days from Mar 6th 2024
- `ts = pd.Series(np.random.randn(len(rng)),rng)` - series with 5 random samples from ϕ
- `ts.tz_localize("UTC")` - localize tz-naive index of a Series or Dataframe to target time zone

6.10. Categoricals

6.10.1. Converting to categorical data

- conversion is performed using `df.astype("dtype")` method
- `df = pd.DataFrame({"id":[1,2,3,4,5,6],"raw_grade":["a","b","b","a","a","e"]})`
- `df["grade"] = df["raw_grade"].astype("category")` - adds categorical column, cats (auto) = ["a","b","e"]
- access to categories is given through the `cat` attribute
- `df["grade"] = df["grade"].cat.rename_categories(new_categories)` - renames existing categories
- `df["grade"] = df["grade"].cat.set_categories(extended_categories)` - adds missing categories
- grouping by categorical column with `observed=False` also shows empty categories:
- `df.groupby("grade", observed=False).size()` - returns categories with number of rows 0 or more

6.11. Importing and exporting data

- importing: `read_FORMAT` functions, exporting: `to_FORMAT` functions

6.11.1. CSV

- `pd.read_csv(file_path, *, sep=',', delimiter=None, header='infer', names=[], index_col=None, usecols=None, skiprows=None, skipfooter=0, nrows=None, skip_blank_lines=True, ...)`
- `sep` = char or regex to treat as delimiter, `header` = row number containing column labels, `names` = sequence of column labels to apply, `index_col` = columns to use as row labels, `usecols` = subset of columns to select, `skiprows` = line numbers to skip or number of lines to skip, `skipfooter` = number of lines from bottom to skip, `nrows` = number of rows to read, ...
- `pd.to_csv(file_path, *, sep=',', columns=None, header=True, index=True, index_label=None, ...)`
- `sep` = field delimiter for the output file, `columns` = columns to write, `header` = write out the column names (if list, assumed to be column aliases), `index` = write row names, `index_label` = col label for index col)

6.12. Example:

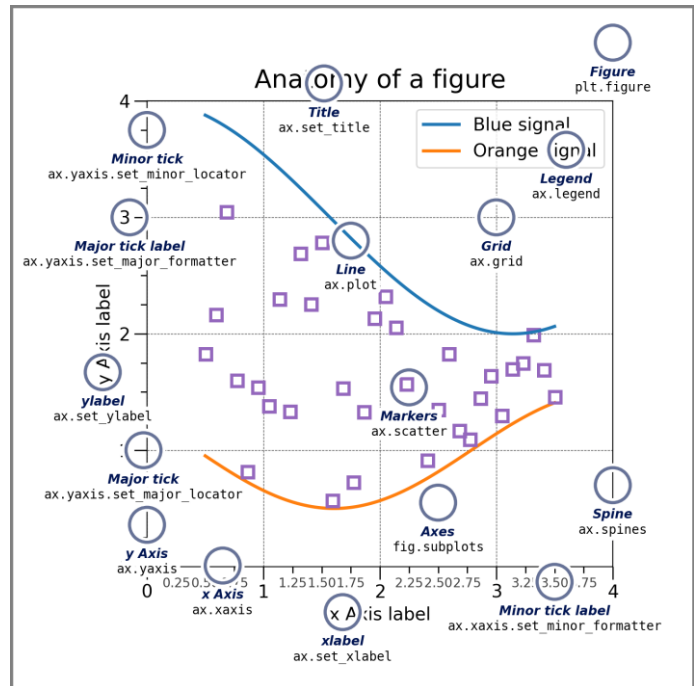
- `import numpy as np`
- `import pandas as pd`
- *# data loading and df creation*
- `red_wine = pd.read_csv(file_name, delimiter=';')`
- `white_wine = pd.read_csv(file_name, delimiter=';')`
- `wine_data = pd.concat([red_wine, white_wine], ignore_index=True)`
- *# data cleaning and preparation*
- `wine_data.replace([float('inf'), float('-inf')], float('nan'), inplace=True)`
- *# creating correlation matrix and getting max correlation*
- `corr_matrix = wine_data.corr()`
- `ph_corr = corr_matrix["pH"].drop("pH")` # otherwise pH would be most correlated with pH
- `ph_corr_max = ph_corr.idxmax()` # row index value of the highest correlation with pH
- *# plotting results*
- `print(f"Most correlated attribute with pH is: {ph_corr_max}")`
- `print(f"Correlation between pH and {ph_corr_max}: {ph_corr[ph_corr_max]}")`

7. Matplotlib

7.1. Basic structure

7.1.1. Hierarchy:

- data is graphed on Figure, each of which can contain one or more Axes, an area where points can be specified in terms of coordinates
- the simplest way of creating a Figure with an Axes is using `pyplot.subplots` - we can then use `Axes.plot` to draw same data on the Axes and show to display the figure
- # create figure containing single Axes
- `fig, ax = plt.subplots()`
- # plot some data on the Axes
- `ax.plot([1,2,3,4],[1,2,3,4])`
- # show figure (if required by backend)
- `plt.show()`

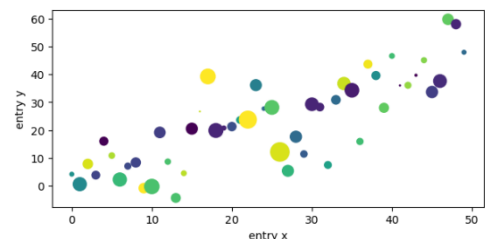


7.1.2. Parts of a Figure

- Figure:** the whole figure. It keeps track of all the child Axes, a group of special Artists (titles, legends, colorbars, etc.) and nested subfigures
- typical functions for creating a Figure:
 - `fig = plt.figure()` - an empty figure with no Axes
 - `fig, ax = plt.subplots()` - a figure with a single Axes
 - `fig, axs = plt.subplots(2,2)` - a figure with 2x2 grid of Axes
 - `fig, axs = plt.subplot_mosaic([['left','right_top'], ['left','right_bottom']])` - 1 Axes on the left, 2 on the right
- Axes:** Artist attached to a Figure that contains a region for plotting data, usually includes 2/3 Axis objects
- each Axes also has a title (set via `set_title()`), and x&y labels (set via `set_xlabel()` and `set_ylabel()`)
- the Axes methods are the primary interface for configuring most parts of the plot (data, scales, labels,...)
- `Axes.plot`: plot y vs x as lines and/or markers, returns a list of Line2D
- Artist:** everything visible on the Figure - Text, Line2D, Patch, ... - even Figure, Axes and Axis
- most Artists are tied to an Axes; such an Artist cannot be shared by multiple Axes, or moved

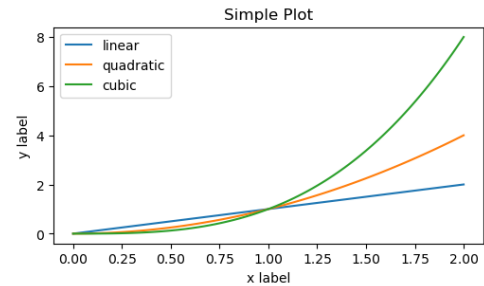
7.1.3. Types of inputs to plotting functions

- expected input is `np.array` or `np.ma.masked_array`, or objects that can be passed to `np.asarray`
- most methods will also parse string-indexable objects like dicts or `pd.DataFrame`:
- `x = np.arange(50)`
- `y = x + 10*np.random.randn(50)`
- `c = np.random.randint(0,50,50)` # c = color
- `s = np.abs(np.random.randn(50))*100` # s = size
- `data = {'x':x,'y':y,'c':c,'s':s}`
- `fig,ax = plt.subplots(figsize=(5,3),layout='constrained')`
- `ax.scatter('x','y',c='c',s='s',data=data)`
- `ax.set_xlabel('entry x')`
- `ax.set_ylabel('entry y')`



7.1.4. Coding style

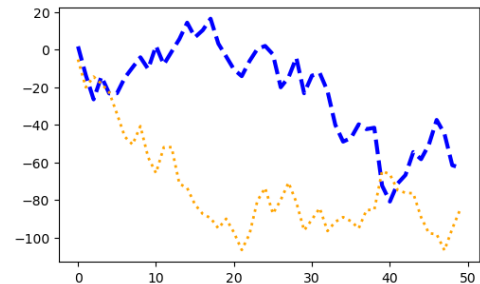
- *Explicit interface coding style ("OO-style")*: explicitly create *Figure* and *Axes* and call methods on them
- `x = np.linspace(0,2,100)`
- `fig,ax = plt.subplots(figsize=(5,3),layout='constrained')`
- `ax.plot(x,x,label='linear')`
- `ax.plot(x,x**2,label='quadratic')`
- `ax.plot(x,x**3,label='cubic')`
- `ax.set_xlabel('x label')`
- `ax.set_ylabel('y label')`
- `ax.set_title('Simple Plot')`
- `ax.legend()`



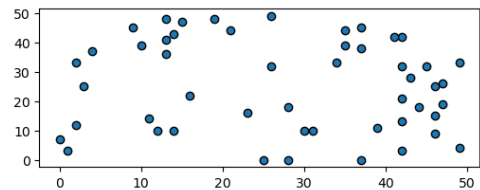
7.2. Styling Artists

- *most methods have styling options, accessible either on method call or from a setter*

- `x = np.arange(50)`
- `y1 = 0.1*x + np.cumsum(np.random.randn(50))*10`
- `y2 = -0.1*x + np.cumsum(np.random.randn(50))*10`
- `fig,ax = plt.subplots(figsize=(5,3),layout='constrained')`
- `ax.plot(x,y2,color='blue',linewidth=3,linestyle='--')`
- `l, = ax.plot(x,y2)`
- `l.set_color('orange')`
- `l.set_linewidth(2)`
- `l.set_linestyle('.')`

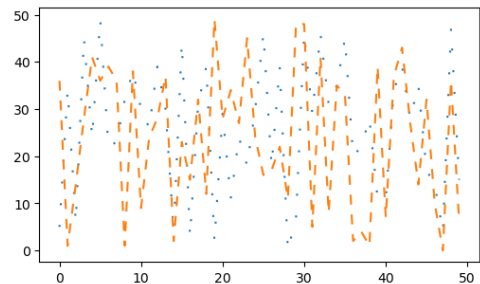


- *some Artists will take multiple colors*
- *for scatter plot the edge of the markers can be different colors from the interior*
- `x = np.random.randint(0,50,50)`
- `y = np.random.randint(0,50,50)`
- `fig,ax = plt.subplots(figsize=(5,3),layout='constrained')`
- `ax.scatter(x, y, s=50, facecolor='C0', edgecolor='k')`

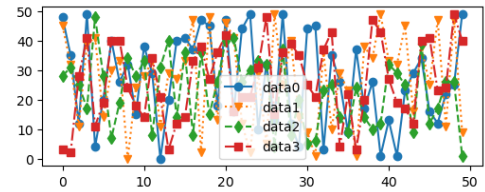


- *linewidths*
- *typically in typographic points (1 pt = 1/72 inch)*
- *available for Artists that have stroked lines*

- *linestyles*
- *named linestyles*: 'solid' ('-'), 'dotted' (':'), 'dashed' ('--'), 'dashdot' ('-.')
- *refined control with (offset,(on_off_seq)) tuple*
- *available for Artists that have stroked lines*
- `x = np.arange(50)`
- `y1 = np.random.randint(0,50,50)`
- `y2 = np.random.randint(0,50,50)`
- `fig,ax = plt.subplots(figsize=(5,3),layout='constrained')`
- `l1, = ax.plot(x,y1)`
- `l2, = ax.plot(x,y2)`
- `line_1pt_space_10pt = (0,(1,10))`
- `l1.set_linestyle(line_1pt_space_10pt)`
- `line_5pt_space_5pt = (0,(5,10))`
- `l2.set_linestyle(line_5pt_space_5pt)`



- markers
- size depends on the method (plot > diameter, in points, scatter > proportional to visual area of marker)
- styles - string codes (".", "o" - circle, "<,v,>,"^" - triangles, "s" - square, "*" - star, "+", "x", "|", "_", "...)
- `x = np.arange(50)`
- `m = ['o','v','d','s']`
- `s = ['-','.', '--', '-']`
- `y = [np.random.randint(0,50,50) for _ in range(len(m))]`
- `fig,ax = plt.subplots(figsize=(5,2),layout='constrained')`
- for `i,t` in `enumerate(y)`:
- `ax.plot(x,t,m[i],label=f"data{i}",linestyle=s[i])`
- `ax.legend()`

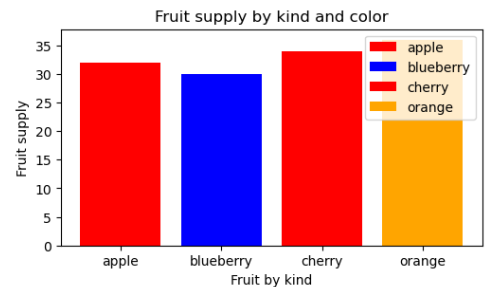


7.3. Graph types

- `Axes.GRAPH_TYPE(...)`:

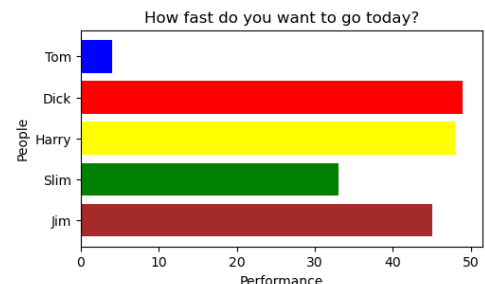
7.3.1. `bar(x,height,width=0.8,,align='center',data=None):`

- `fruits = ['apple','blueberry','cherry','orange']`
- `counts = np.random.randint(0,50,len(fruits))`
- `colors = ['red','blue','red','orange']`
- `fig,ax=plt.subplots(figsize=(5,3),layout='constrained')`
- `ax.bar(fruits,counts,color=colors)`
- `ax.set_xlabel('fruit by kind')`
- `ax.set_ylabel('fruit supply')`
- `ax.set_title('Fruit supply by kind and color')`



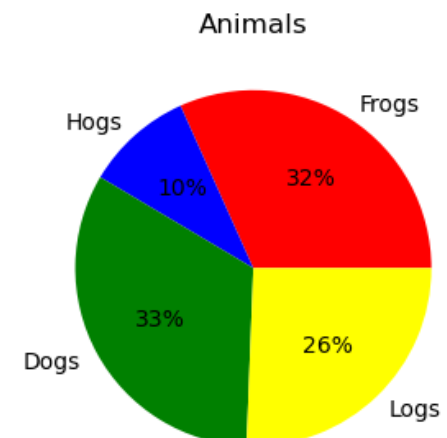
7.3.2. `barh(y,width,height =0.8,align='center',data=None):`

- `people = ['Tom','Dick','Harry','Slim','Jim']`
- `performance = np.random.randint(0,50,len(people))`
- `colors = ['blue','red','yellow','green','brown']`
- `fig,ax=plt.subplots(figsize=(5,3),layout='constrained')`
- `ax.barh(people,performance,color=colors)`
- `ax.invert_axis()` # default is stacking bottom to top
- `ax.set_ylabel("People")`
- `ax.set_xlabel("Performance")`
- `ax.set_title("How fast do you want to go today?")`



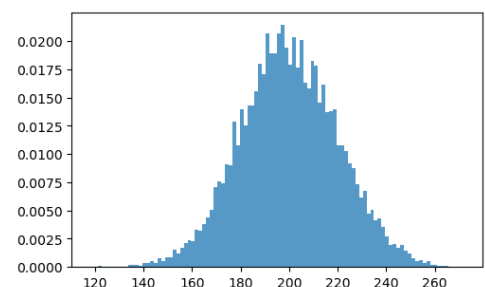
7.3.3. `pie(x,labels=None,colors=None,autopct=None...):`

- `label = ["Frogs","Hogs","Dogs","Logs"]`
- `size = np.random.randint(0,50,len(labels))`
- `color = ["red","blue","green","yellow"]`
- `fig,ax = plt.subplots(figsize=(5,3),layout='constrained')`
- `ax.pie(size,labels=label,colors=color,autopct='%0.0f%%')`
- `ax.set_title("Animals")`



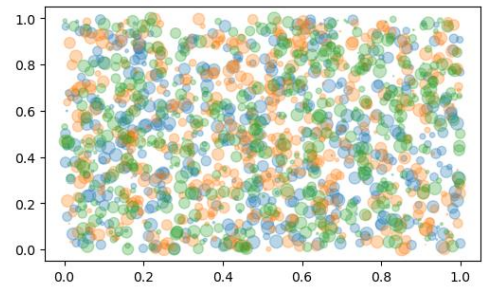
7.3.4. `hist(x,bins=None,density=True,histtype='bar',alpha...):`

- # density = adjust so that area under hist integrates to 1
- # alpha = bins transparency (0 = invisible, 1 = opaque)
- `x = np.random.normal(200,25,size=10000)`
- `fig,ax = plt.subplots(figsize=(5,3),layout='constrained')`
- `ax.hist(x,bins=20,density=True,alpha=0.75)`



7.3.5. `scatter(x,y,s=None,c=None,marker=None,alpha...)`

- `colors = ['tab:blue','tab:orange','tab:green']`
- `n = 750`
-
- `fig,ax = plt.subplots(figsize=(5,3),layout='constrained')`
- for color in colors:
- `x = np.random.rand(n)`
- `y = np.random.rand(n)`
- `scale = 100*np.random.rand(n)`
- `ax.scatter(x,y,c=color,s=scale,label=color,alpha=0.3)`



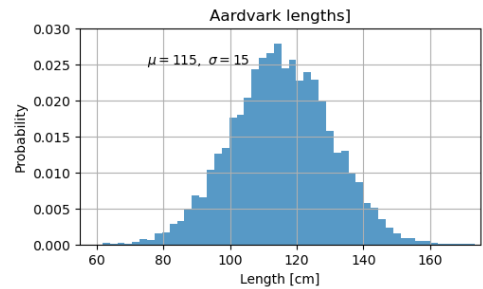
7.4. Labelling plots

7.4.1. basic labels:

- `xlabel`, `ylabel`, `text`, `title`, `grid`, `legend`

7.4.2. histogram example:

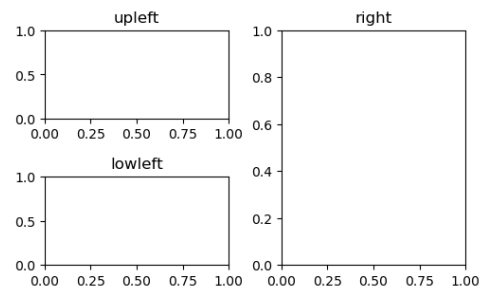
- `mu,sigma = 115,15`
- `x = mu + sigma*np.random.randn(10000)`
-
- `fig,ax = plt.subplots(figsize=(5,3),layout='constrained')`
- `n,bins,patches = ax.hist(x,50,density=True,alpha=0.75)`
- `ax.set_xlabel('Length [cm]')` # add text on x label
- `ax.set_ylabel('Probability')` # add text on y label
- `ax.text(75,.025,r'$\mu=115,\sigma=15$')` # add text
- `ax.set_title('Aardvark lengths')` # add the plot title
- `ax.axis([55,175,0,0.03])` # xmin/max,ymin/max
- `ax.grid(True)` # make grid visible



7.5. Working with multiple Figures and Axes

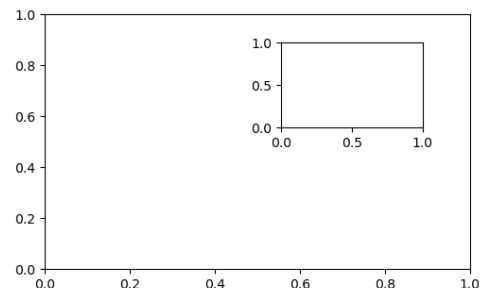
7.5.1. mosaic

- `fig,axd = plt.subplot_mosaic([['upleft','right'], ['lowleft','right']])`
- `figsize=(5,3),layout='constrained')`
-
- `axd['upleft'].set_title('upleft')`
- `axd['lowleft'].set_title('lowleft')`
- `axd['right'].set_title('right')`



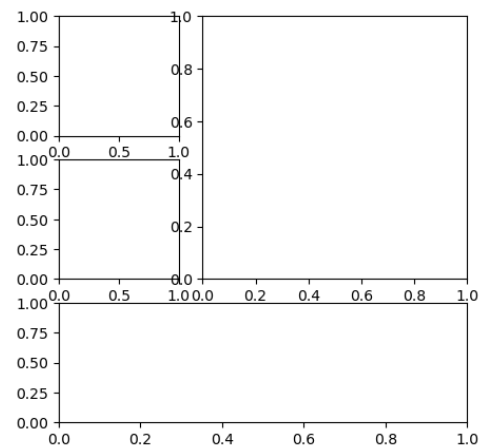
7.5.2. dynamic Axes addition

- `fig = plt.figure(figsize=(5,3))`
-



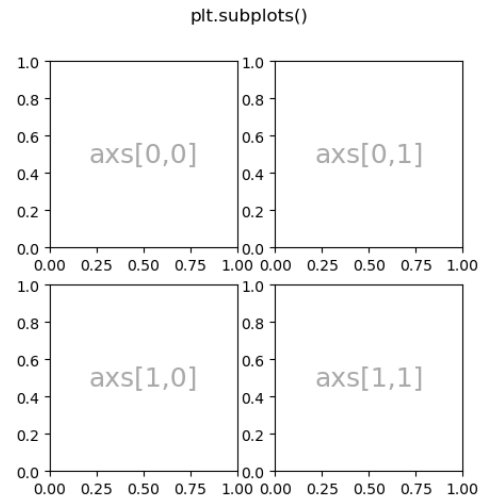
7.5.3. gridspec and dynamic subplot addition

- `fig = plt.figure(figsize=(5,3))`
-
- # add_axes rectangle [L,B,W,H]
- `ax1 = fig.add_axes([.1,.1,.9,.9])`
- `ax2 = fig.add_axes([.6,.6,.3,.3])`
-
- # add_gridspec(nrows=1,ncols=1,...)
- `gs = fig.add_gridspec(3,3)`
-
- # add_subplot(
- `fig.add_subplot(gs[0,0])`
- `fig.add_subplot(gs[1,0])`
- `fig.add_subplot(gs[2,1:])`
- `fig.add_subplot(gs[2,:])`



7.5.4. basic grid using subplots

- `fig,axs = plt.subplots(nrows=2,ncols=2)`
- `for row in range(2):`
- `for col in range(2):`
- `axs[row,col].annotate(f'axs[{row},{col}]',`
- `(0.5,0.5),transform=axs[row,col].transAxes,`
- `ha='center',va='center',fontsize=18,`
- `color='darkgrey')`
- `fig.suptitle('plt.subplots()')`



7.6. Plotting using Pandas

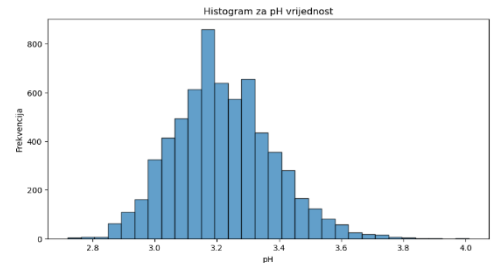
- `df.plot` . makes plots of Series or DataFrame
- by default matplotlib backend is used (`plotting.backend`)
-

7.6.1. parameters:

- `data`: object for which the method is called
- `x=None`: label or position (only for `df`)
- `y=None`: label or list of labels, allows plotting one col vs another (only for `df`)
- `kind = {'line' | 'bar' | 'barh' | 'hist' | 'box' | 'kde' | 'density' | 'area' | 'pie' | 'scatter' | 'hexbin'}`
- `ax=None`, an axes of the current figure (matplotlib axes object)
- `subplots=False`: if True, make separate subplots for each column
- `layout`: tuple (rows, cols) for the layout of the subplots
- `use_index=True`: use index as ticks for x axis
- `title`: title to use for the plot (str for single or list of str for subplots)
- `grid=False`: if True, plot axis grid lines
- `legend=False`: if True, place legend on axis subplots
- `xticks,yticks=sequence`: values to use for the ticks
- `xlabel,ylabel=None`: label to use for the x and y axis
- `rot=None`: rotation (float) for ticks
- `fontsize=None`: float, font size for ticks

7.6.2. Example (wine continued):

- `fig,ax = plt.subplots(figsize=(10,5),layout='constrained')`
- `ax.hist(wine_data['pH'].dropna(),bins=30,alpha=0.75)`
- `ax.set_title("Histogram for pH value")`
- `ax.set_xlabel('pH')`
- `ax.set_ylabel('Frequency')`



7.6.3. Example (wine continued):

- `fig,ax = plt.subplots(figsize=(10,5),layout='constrained')`
- `ax.hist(wine_data[ph_corr_max].dropna(),bins=30,alpha=0.75)`
- `ax.set_title(f"Histogram for most correlated attribute ({ph_corr_max})")`
- `ax.set_xlabel({ph_corr_max})`
- `ax.set_ylabel('Frequency')`

