



Distance Learning System

# Ugrađeni objekti

Object Oriented Programming Python

# Enkapsulacija, geteri i seteri

- Enkapsulacija je sakrivanje logike nekog objekta od njegovog korisnika
- Implementacija ovog sistema najčešće podrazumeva, takozvane **getere** i **setere**
- Geter-i i seter-i su metode koje se koriste za kontrolisano pristupanje privatnim poljima klase
- Polja koja su izložena kroz getere i setere, nazivaju se svojstva / properties

Klasa Point ima "najbolju nameru" da čuva poziciju i menja njene koordinate

```
class Point:
    def __init__(self):
        self.a = 0
        self.b = 0
    def move(self):
        self.a+=1
        self.b+=1
```

Ipak, nepravilnom upotrebom, klasa može izazvati probleme zbog izloženosti ključnih polja

```
pt = Point()
pt.b = "Hello"
pt.move()
```

Ovo je koska

# Geteri i seteri (java way)

- Polja klase možemo zaštititi tako što ćemo da ih učinimo privatnim, a njihov pristup omogućimo pomoću metoda

Ovakve metode, nazivaju se **geteri** i **seteri**

```
class Point:
    def __init__(self):
        self.__a = 0
    def get_a(self):
        return self.__a
    def set_a(self,a):
        if not isinstance(a,int):
            return
        self.__a = a
    def move(self):
        self.__a+=1
```

- Ovo sada ne izaziva poremećaj u radu programa

```
pt = Point()
pt.set_a("Hello")
pt.move()
```

- Ovo radi očekivano

```
pt = Point()
pt.set_a(12)
pt.move()
```

# Property dekorator

- Dekorator **property** omogućava elegantniju implementaciju getera i setera
- Dekorator property se obično pojavljuje u kombinaciji sa pratećim dekoratorima (**setter** i **deleter**)

```
class Point:
    __a = 5

    @property
    def a(self):
        return self.__a

    @a.setter
    def a(self, val):
        if not isinstance(val, int):
            return
        self.__a = val

pt = Point()
pt.a = 10
print(pt.a)
```

# Vežba - Card (oopp-ex03 card.py)

- Potrebno je kreirati Python klasu kojim će biti predstavljena karta
- Klasa treba da ima kao svojstva broj karte, boju i šifru (kod) karte
- Boja se može označiti stringom ili celobrojnomo vrednošću
- Šifra karte je string



# Ugrađeni Python moduli (Python Standard Library)

---

- Standardna Python biblioteka je sastavni deo Python instalacije i sadrži mnoštvo različitih modula
- Iako moduli iz standardne Python biblioteke nisu sastavni deo jezgra jezika, njihovo poznavanje je esencijalno za programiranje u ovom jeziku
- Češće korišćeni moduli će biti predstavljeni u nastavku, ali je snalaženje u dokumentaciji neophodno za ozbiljniji rad



*<https://docs.python.org/3.8/library/index.html>*

# Modul os

<https://docs.python.org/3/library/os.html>



- Modul izlaže funkcionalnosti operativnog sistema na kome se Python izvršava
- Veoma je važan za sistemsko programiranje
- Dobar je zato što možemo interoperabilno da manipulišemo sistemskim funkcionalnostima
- Omogućava rad na niskom nivou sa: procesima, fajlovima, soketima i slično
- Os modul takođe daje na raspolaganje dosta informacija o sistemu na kome se izvršava. Na primer, broj procesora, količina memorije

# Modul os - primeri upotrebe

<https://docs.python.org/3/library/os.html>

---

- Preimenovanje fajla / direktorijuma:  

```
os.rename("mydir1", "mydir2")
```
- Dobijanje apsolutne putanje od relativne  

```
os.path.abspath("./")
```
- Svi fajlovi / direktorijumi jednog direktorijuma  

```
for f in os.scandir("./"): print(f.name)
```



# Modul sys

<https://docs.python.org/3/library/sys.html>

---



- Ovaj modul sadrži podatke i operacije vezane za Python interpreter koji izvršava aktuelnu skriptu
- Pomoću modula sys, možemo videti detalje o skripti, preuzeti ulazne parametre skripte, zaustaviti skriptu, startovati eksterni sistemski proces

# Modul sys

<https://docs.python.org/3/library/sys.html>



- Provera operativnog sistema (oopp-ex03 otest.py)  

```
if sys.platform == 'linux2': print("Computer runs Linux")
```
- Preuzimanje parametara startovanja (oopp-ex03 calc.py)  

```
print(int(sys.argv[1])+int(sys.argv[2]))
```
- Prikaz učitanih modula (oopp-ex03 modules.py)  

```
for mod in sys.modules: print("Module: " , mod)
```

# Rad sa vremenom (modul time)

<https://docs.python.org/3/library/time.html>

---

- Python poznaje više ugrađenih modula za rad sa vremenom (time, datetime, calendar)
- Osnovni modul za rad sa vremenom je modul **time**
- Modul time je port C biblioteke za rad sa vremenom pa su nam njegove funkcije eventualno poznate i iz drugih jezika čiji su prevodioci napisani u jeziku C

```
import time
```

# Funkcija sleep

---

- Funkcija sleep pauzira program na određeno vreme izraženo u sekundama, ali sa mogućnošću unošenja decimalnih vrednosti (dakle, pauza može biti i manja od sekunde)
- Ovaj način pauziranja je dobar za linearno izvršavanje, ali se prilikom rada sa nitima mora pažljivo koristiti jer nije sinhronizovan
- Sledeći kod ispisuje poruku Hello, a zatim, nakon dve sekunde ispisuje poruku world

```
print("Hello...")  
time.sleep(2)  
print("...world")
```

# Vežba Reminder (oopp-ex03 reminder.py)

- Korisnik unosi broj sekundi za timer i poruku
- Nakon unosa, startuje se odbrojavanje i prikazuje se koliko još ima vremena do isteka
- Po završenom odbrojavanju, prikazuje se poruka deset puta, a zatim se program završava

```
Enter message: Stop playing WoW!!!
Enter time: 3
Time remaining: 1
Stop playing WoW!!!
Stop playing WoW!!!
Stop playing WoW!!!
Stop playing WoW!!!
Stop playing WoW!!!
Stop playing WoW!!!
Stop playing WoW!!!
Stop playing WoW!!!
Stop playing WoW!!!
Stop playing WoW!!!
```

# Funkcija time

- Funkcija **time** vraća broj sekundi od referentne tačke
- Referentna tačka je u Python-u, za time funkciju, 1. januar 1970 godine
- Na osnovu referentne tačke i broja sekundi od broja sekundi koliko je od nje prošlo, moguće je izračunati bilo koji datum
- U zavisnosti od platforme, mogu se pojaviti i frakcije sekunde

```
import time  
print(time.time())
```

- Broj sekundi  
od 1.1.1970

- Broj mikrosekundi  
od poslednje  
sekunde

1574461153.94317


# Rukovanje milisekundama

- Jedino što nam je zapravo potrebno da bi radili sa vremenom jeste trenutni broj milisekundi koje su protekle od 01.01.1970. godine
- U Pythonu imamo mogućnost da dobavimo ovu informaciju na više načina
- Obzirom da funkcija `time` ima decimalni zarez, kome uvek prethodi broj sekundi, možemo reći da su iza zareza frakcije sekunde
- Množenjem ovog broja sa 1000 (jer se sekunda sastoji od 1000 milisekundi) dobićemo relativan broj milisekundi u odnosu na referentnu tačku

```
millis = round(time.time() * 1000)
```

- Naredba daje rezultat poput sledećeg: *1415559852660*
- Sledeći kod prikazuje broj milisekundi u mrtvoj petlji

```
while True:  
    millis = round(time.time() * 1000)  
    print(millis)
```



```
1574511322944  
1574511322944  
1574511322944  
1574511322945  
1574511322945  
1574511322945  
1574511322945  
1574511322945  
1574511322945
```

# Funkcija `gmtime` / `localtime`

- Vraća trenutno vreme u odnosu na referentnu tačku
- Vraćena vrednost je strukturnog tipa (named tuple) i sadrži pojedinačne komponente trenutnog datuma i vremena

```
gmt = time.gmtime()
print(f"Current date is: {gmt.tm_mday} {gmt.tm_mon} {gmt.tm_year}")
```

- Podrazumevano, `gmtime` generiše vreme na osnovu referentne tačke (implicitnim pozivom `time` funkcije), ali takođe, vreme se može generisati i na osnovu eksplicitne vrednosti
- Na primer, sledeći kod će generisati vremenski objekat za datum 1.1. 1990 godine, što je godina kada je stvoren Python

```
gmt = time.gmtime(631152000)
```

- Funkcija **`localtime`** se ponaša identično kao i funkcija `gmtime`, samo što, umesto univerzalnog, prikazuje vreme za zonu koja je podešena na računaru



# Funkcija `ctime` / `asctime`

---

- **`ctime`** funkcioniše isto kao i `gmtime/localtime`, samo što vraća vreme u formatiranom stringu umesto u tuple-u

```
print(time.ctime(631152000))
```



```
Mon Jan 1 01:00:00 1990
```

- **`asctime`** radi kao i `ctime`, ali umesto timestamp-a, kao parametar prihvata vremenski tuple (`gmtime` ili `localtime`)

```
date = time.localtime(631152000)  
print(time.asctime(date))
```



```
Mon Jan 1 01:00:00 1990
```

# Funkcije strftime i strptime

- **strftime** formatira datum po korisnički definisanom šablonu
- **strptime** generiše time strukturu od stringa, po definisanom šablonu

```
date = time.localtime(631152000)
print(time.strftime("Month: %b Day: %d Year: %Y",date))
```



Month: Jan Day: 01 Year: 1990



```
date = time.strptime('Jan 01 1990', '%b %d %Y')
print(date)
```

- Za parametara formatiranja (%b,%d,%Y...) treba konsultovati dokumentaciju:

<https://docs.python.org/3/library/time.html#time.strftime>

# Modul Calendar

<https://docs.python.org/3/library/calendar.html>


- Ovaj modul većinom se bavi kalendarskim elementima vremena: danima u nedelji, mesecima, da li je godina prestupna ili ne i slično
- Pogodan je za tekstualno formatiranje kalendara, ali i kao alat za manipulaciju
- Modul čini klasa Calendar, i različite prateće klase većinom namenjene formatiranju

*Svi dani tekućeg meseca, po nedeljama*

```
cal = calendar.Calendar()
tm = time.localtime()
all_dates = cal.monthdatescalendar(tm.tm_year, tm.tm_mon)
for week in all_dates:
    for day in week:
        print(day)
```

*Formatiran tekući mesec*

```
tm = time.localtime()
print(calendar.month(tm.tm_year, tm.tm_mon))
```



| November 2019 |    |    |    |    |    |    |
|---------------|----|----|----|----|----|----|
| Mo            | Tu | We | Th | Fr | Sa | Su |
|               |    |    |    | 1  | 2  | 3  |
| 4             | 5  | 6  | 7  | 8  | 9  | 10 |
| 11            | 12 | 13 | 14 | 15 | 16 | 17 |
| 18            | 19 | 20 | 21 | 22 | 23 | 24 |
| 25            | 26 | 27 | 28 | 29 | 30 |    |

# Vežba FPS Counter (oopp-ex03 fpscounter.py)

---

- Potrebno je kreirati program koji će izvršavati određenu funkciju u određenom vremenskom intervalu (na primer 30 puta u sekundi)
- Funkcija koji treba pozivati se može zvati **tick**
- Unutar funkcije tick, treba prikazati trenutnu brzinu izvršavanja (broj frejmova po sekundi)



Fps: 60

# Rad sa modulom datetime

<https://docs.python.org/3.8/library/datetime.html>

---


- Datetime je modul koji sadrži različite funkcionalnosti vezane za vreme
- Datetime koristimo kada želimo jednostavno da izvršimo različite operacije nad vremenom:
  - Da poredimo vreme
  - Da dodamo vreme na postojeće ili oduzmemo od postojećeg
  - Da izračunamo interval ili period
  - Da formatiramo vreme iz stringa ili u string
  - Da radimo samo sa datumom ili samo sa vremenom

# Klasa datetime

---

- Klasa datetime je glavna klasa ovog modula
- Klasu možemo instancirati pomoću konstruktora, kada moramo proslediti sve vrednosti neophodne za njeno funkcionisanje (minimalno dan, mesec i godina) ili preko neke od klasnih metoda

```
dt1 = datetime.datetime(2019,11,23,15,38,46)
dt2 = datetime.datetime.now()
dt3 = datetime.datetime.fromtimestamp(time.time())
print(dt1)
print(dt2)
print(dt3)
```



```
2019-11-23 15:38:46
2019-11-23 15:40:18.612380
2019-11-23 15:40:18.612388
```

# Korisničko definisanje vrednosti

- Datetime objekat može se takođe dobiti formatiranjem stringa
- Prilikom formatiranja, koriste se metode istoimene funkcijama time modula (strftime i strptime)

```
dt1 = datetime.datetime.strptime('23 11 2019', '%d %m %Y')  
print(dt1.strftime('%d %b %Y'))
```



```
23 Nov 2019
```

# Vežba GpsParse (oopp-ex03 gpsparse.py)

---

- U aplikaciju ulaze podaci sa gps uređaja
- Za svaku tačku, gps uređaj šalje sledeću poruku:
- 22052014,44.756364,20.412598,051230,123143124122
- Podaci u poruci su (respektivno): datum, latituda, longituda, vreme i imei uređaja
- Potrebno je kreirati klasu koja je u stanju da parsira podatke i čuva informacije o svim primljenim podacima u smislenim tipovima
- Iako većina uređaja šalje poruke kao što je ova u primeru, neki uređaji šalju poruke u kojima datum i vreme imaju izmenjene pozicije. Ovakvi uređaji šalju još jedan dodatni podatak (oznaku H) na kraju poruke:
- (051230,44.756364,20.412598,22052014,123143124122,H)



# Rukovanje periodom (timedelta)

<https://docs.python.org/3.8/library/datetime.html#datetime.timedelta>

- Osim formatiranja, prilikom rukovanja vremenom, naročito je važno znati odrediti period između dve vremenske jedinice
- Ovo se u datetime modulu, predstavlja klasom **timedelta**
- Klasa timedelta može se dobiti eksplicitno, instanciranjem, ili implicitno, izvršavanjem neke operacije nad dva datuma

```
start = datetime.datetime.strptime("10 11", "%d %m")  
end = datetime.datetime.strptime("13 11", "%d %m")  
period = start - end  
print(period)
```



```
3 days, 0:00:00
```

# Dodavanje i oduzimanje vremena

---

- Datetime objektu možemo izmeniti datum dodavanjem, odnosno, oduzimanjem određenog perioda
- Ovo se takođe postiže klasom timedelta

```
today = datetime.datetime.now()
p = datetime.timedelta(2,0,0,0,0,0,0)
day_after_tomorrow = today + p

print(today)
print(day_after_tomorrow)
```



```
2019-11-23 19:30:57.262651
2019-11-25 19:30:57.262651
```

# String <https://docs.python.org/3.8/library/string.html>

---

- string ima neke osobenosti na koje moramo obratiti pažnju tokom rada ako hoćemo da budemo efikasni
  - string je **nemutabilan** – ne može mu se menjati sadržaj
  - string možemo kreirati jednostavnom dodelom: **word = "Hello!"**
- Ono što ćemo najčešće želeći od stringa, jeste preuzimanje njegovog sadržaja, njegovo formatiranje, sastavljanje više stringova u jedan
- Osim string tipa (str) Python poznaje i string modul, koji sadrži pomagala za rad sa stringovima

# Konkatinacija stringova

- Konkatinacija stringa je zapravo dodavanje jednog stringa na drugi, čime se dobija novi
- Konkatinaciju možemo uraditi na više načina, ali su tri najčešća:
  - Korišćenjem operatora **+**
  - Korišćenjem metode **join**
  - Formatiranjem, pomoću operatora **%**
  - Formatiranjem pomoću metode **format**
  - Formatiranjem pomoću **f-string**-a

*print("This presenter is !!!")*



# Konkatinacija pomoću operatora + i metode join

- Metod **+** od dva stringa, pravi treći string

```
sentence = "Hello " + "World"  
print(sentence)
```

- Ovaj način spajanja je tipski osetljiv, pa nije moguće praviti kombinacije stringa sa nekim tipovima koji to nisu. Na primer, sledeći kod će izazvati grešku

```
sentence = "Hello " + 3  
print(sentence)
```

- Metod **join** se može smatrati operacijom nad nizovima, iako je deo string-a. Ovaj string metod spaja sve članove niza u jedan string

```
sentence = "".join(["Hello", " world", ",", " how", " are", " you"])  
print(sentence)
```

# Konkatinacija stringa pomoću formatera

- Postoji nekoliko načina za formatiranje stringa
  - **%-string** i metod **format** predstavljaju zapravo stariju i noviju varijantu istog formatiranja

Kod %-string-a, tekst sa placeholderima se odvaja od parametara oznakom %

Parametri su u formi tuple-a

```
sentence = "%s %s" % ('Hello ', 'world')  
print(sentence)
```

Kod format metode, tekst sa placeholderi se označavaju vitičastim zagradama i nema tipizacije

Parametri su u formi tuple-a

```
sentence = "{} {}".format('Hello ', 'world')  
print(sentence)
```

Parametri takođe mogu biti pozicionirani, pa čak i dodatno formatirani

```
sentence = "{} {:.2f}".format('Speed: ', 1.1234)  
print(sentence)
```

# f-string

---

- F-string je formatiranje u kome se unutar samog stringa nalaze dinamički delovi (ne placeholderi)
- Svaki deo koji treba evaluirati, označava se vitičastim zagradama

```
x = 10
y = 20
z = x + y
print(f"{x} + {y} = {z:.2f}")
```

# Template

---

- Klasa Template modula string, omogućava umetanje elemenata u postojeći tekst
- Ova varijanta je idealna za kreiranje dinamičkih HTML sadržaja
- Placeholderi se predstavljaju oznakom **\$**
- Ključna reč placeholdera se zatim menja za unos prosleđenog rečnika

```
d = {"title": "Hello world"}  
text = "<h3>$title</h3><p>How are you</p>"  
tmp = string.Template(text)  
print(tmp.substitute(d))
```



```
<h3>Hello world</h3><p>How are you</p>
```



# Vežba Tank (oopp-ex03 tank.py)

---

- Potrebno je kreirati program koji generiše html stranu za prikaz tenka
- Korisnik na početku programa unosi naziv tenka, adresu do strane sa tenkom i adresu slike tenka.
- Program na osnovu unetih podataka treba da generiše html stranu koja ima sledeći izlaz:

Html: <https://bitbucket.org/snippets/vmaric/zeX8bB>



# Rad sa nizovima karaktera

---

- Vrednost tipa string je takođe i niz stringova dužine jednog karaktera.
- Dužinu ovog niza možemo dobiti funkcijom **len()**, a određenog člana **indeks** operacijom


```
message = "Hello from my string"
print(len(message))
for c in message:
    print(c)
```

```
print(f"Character at index 4 is {message[4]}")
```

# Konverzija

- Stringove često pretvaramo u brojeve i obrnuto
- Da bismo pretvorili string u određeni numerički tip, koristimo konstruktor tog tipa

```
a = "1"  
print(a + a)  
a = int(a)  
print(a + a)
```



```
11  
2
```

- Prilikom konverzije, string mora sadržati ispravan materijal za konverziju

Neispravno

```
txt = "a"  
a = int(txt)
```

Ispravno

```
txt = "125"  
a = int(txt)
```

# Konverzija

- String objekat sadrži mnoštvo metoda kojima možemo proveriti tip njegove vrednosti, pre konverzije

```
txt = "125"
txt.is
```

- `isalnum`
- `isalpha`
- `isascii`
- `isdecimal`
- `isdigit`
- `isidentifier`
- `islower`
- `isnumeric`
- `isprintable`
- `isspace`
- `istitle`
- `isupper`

- Sa ovakvom konverzijom, (skoro) potpuno smo bezbedni

```
txt = "abc"
a = int(txt) if txt.isnumeric() else 0
```

- Prilikom konverzije možemo birati različite notacije

```
a = int("10",2)
b = int("10",10)
c = int("10",16)
print(a)
print(b)
print(c)
```

→

|    |
|----|
| 2  |
| 10 |
| 16 |

# Enkodiranje, dekodiranje

- Veoma često, kompleksan tip hoćemo da pretvorimo u string, ili string u kompleksan tip
- Ovde prepisivanje `__str__` metode najčešće ne pomaže jer informacije nisu struktuirane niti kompletirane
- String metodama `encode` i `decode` pretvaramo string u niz bajtova i obrnuto
- Enkodiranje i dekodiranje se radi često i sa drugim formatima (JSON, XML i slično)

```
msg = "Hello"  
bts = msg.encode("utf-8")  
for b in bts:  
    print(b)
```



```
72  
101  
108  
108  
111
```

```
bts = bytes((72,101,108,108,111))  
msg = bts.decode("utf-8")  
print(msg)
```



```
Hello
```

# Replace i Split

- Metod `replace` menja sadržaj stringa za neki drugi sadržaj, i kao rezultat vraća novi string (dakle ne radi na postojećem stringu)

```
message = "Hello from my string"  
message = message.replace("my", "your")  
print(message)
```

Hello from your string

- Metod `split` kreira niz stringova od zadatog stringa, po zadatom delimiteru:

```
message = "Hello from my string"  
string_arr = message.split(" ")  
for word in string_arr:  
    print(word)
```

Hello  
from  
my  
string

# Vežba (oopp-ex03 userfromstring)

- Data je sledeća klasa:

```
class User:  
    def __init__(self):  
        self.id = ""  
        self.name = ""  
        self.score = ""
```

- Zna se da će u aplikaciju stizati podaci u sledećem formatu:

id → 1-peter-150 ← score  
          ↑  
          name

- Potrebno je kreirati metod **parse**, unutar klase User. Ovaj metod će na osnovu ulaznog stringa, kreirati objekat tipa User.

# Vežba (oopp-ex03 parsejsonman)

- U aplikaciji koja obrađuje serverski deo online multiplayer igre Date su dve klase:

- Klasa UserPoint

```
class UserPoint:  
    x = 0.0  
    y = 0.0
```

- I klasa UserPosition

```
class UserPosition:  
    userid = 0  
    userpoint = None
```

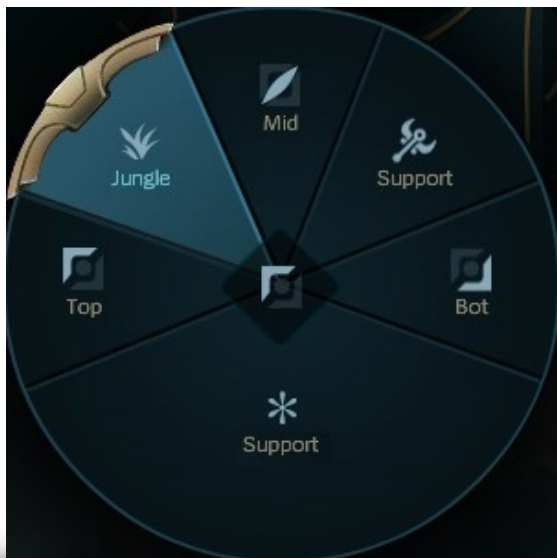
- U aplikaciju u intervalima stiže sledeći string koji predstavlja trenutne pozicije igrača:  
***[{id:10,x:10,y:20},  
{id:5,x:30,y:40},{id:2,x:2,y:7}]***
- Potrebno je izvući smislene podatke iz dobijenog stringa, i smestiti ih u odgovarajuća polja postojećih klasa (objekata), a zatim sve objekte smestiti u listu
- Listu prikazati na izlazu

```
UserId: 10, x: 10, y: 20  
UserId: 5, x: 30, y: 40  
UserId: 2, x: 2, y: 7
```



# Enumeracije

- Enumeracija predstavlja posebnu strukturu čiji je zadatak isključivo skladištenje konstanti nekog tematskog konteksta



# Zašto enumeracija?

- Na primer, ako bi imali klasu Game, hteli bi na neki način da predstavimo stanja te igre (recimo da postoji tri stanja: running, paused i stopped), stanja te igre bi mogla biti predstavljena na sledeći način, pomoću polja klase:

```
class Game:  
    RUNNING = 1  
    PAUSED = 2  
    STOPPED = 3
```

- Zatim bi mogli koristiti kreirana polja na sledeći način:



```
game_status = Game.RUNNING  
  
if game_status == Game.STOPPED:  
    print("Game is stopped")  
elif game_status == Game.PAUSED:  
    print("Game is paused")  
elif game_status == Game.RUNNING:  
    print("Game is running")  
else:  
    print("Who knows?")
```

- Problem kod predstavljenog primera je što je i dalje radimo sa celobrojnim vrednostima

# Implementacija enumeracije

- Seriju polja kojima je predstavljen status igre u prethodnom primeru, pomoću enumeracije možemo definisati na sledeći način:

```
import enum
class GameState(enum.Enum):
    RUNNING = 1
    PAUSED = 2
    STOPPED = 3
```

- I kasnije koristiti identično poljima iz prethodnog primera

```
game_status = GameState.RUNNING

if game_status == GameState.STOPPED:
    print("Game is stopped")
elif game_status == GameState.PAUSED:
    print("Game is paused")
elif game_status == GameState.RUNNING:
    print("Game is running")
else:
    print("Who knows?")
```

# Dobavljanje enumeracije na osnovu stringa ili broja

- U primeru smo videli da je enumeracija dobar način za predstavljanje nekog stanja. Često, ovo stanje ćemo hteti da učinimo perzistentnim (da ga sačuvamo u fajlu ili bazi podataka), i tada moramo sačuvati string ili broj
- Ako enumeraciju čuvamo u obliku stringa ili broja, možemo je kasnije rehabilitovati, kao u sledećim primerima:

```
persisted_state = "RUNNING"  
recovered_state = GameState[persisted_state]  
print(recovered_state)
```

Podatak koji je stigao iz sistema koji ne zna za enumeraciju

Konverzija ulaznog podatka (stringa) u enumeracioni tip

Regularno korišćenje enumeracije

```
persisted_state = 2  
recovered_state = GameState(persisted_state)  
print(recovered_state)
```

Podatak koji je stigao iz sistema koji ne zna za enumeraciju

Konverzija ulaznog podatka (int) u enumeracioni tip

Regularno korišćenje enumeracije

# Izlistavanje enumeracije

---

- Enumeracija ima ugrađen sistem za iteraciju kroz vrednosti
- Ovo nam omogućava da vidimo sve vrednosti koje su nam na raspolaganju unutar enumeracije

```
for g in GameState:  
    print(f"Enum name: {g} , enum value {g.value}")
```

```
Enum name: GameState.RUNNING , enum value 1  
Enum name: GameState.PAUSED , enum value 2  
Enum name: GameState.STOPPED , enum value 3
```

# Proširivanje enumeracionih objekata

- Svaka enumeracija je u stvari objekat
- Definicija ovog objekta vrši se u telu klase enumeracije

```
class GameState(enum.Enum):  
    RUNNING = 1  
    PAUSED  = 2  
    STOPPED = 3  
    def __str__(self):  
        return f"Name: {self.name}, Value: {self.value}"
```



```
print(GameState.RUNNING)
```



```
Name: RUNNING, Value: 1
```

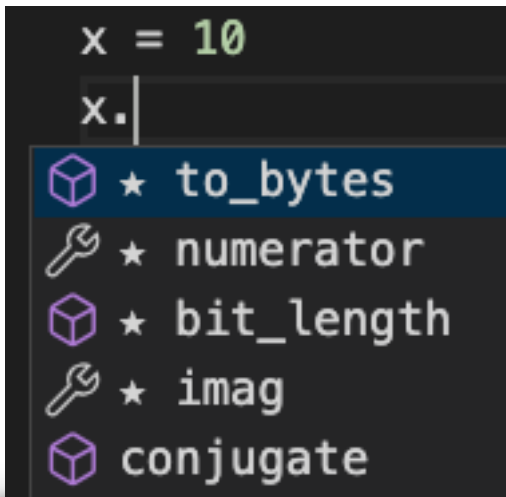
# Zadatak



- Pokušajte da, na što bolji način pomoću enumeracije realizujete listu tipova borbe sa leve strane

# Prosti tipovi podataka

- Svaka prosta vrednost u Python-u, zapakovana je u objekat
- Ovi objekti poseduju različite alate vezane za zapakovani tip





# Inicijalizacija prostih tipova

- Za razliku od ostalih objekata, wrapper prostog tipa možemo kreirati običnom dodelom (bez eksplicitnog instanciranja)
- Takođe, tokom postojanja, objekat će se ponašati kao standardan prosti tip:

```
x = 10
x += 25
print(x)
```

- Objekat možemo takođe inicijalizovati pomoću konstruktora, pri čemu možemo direktno uneti inicijalnu vrednost
- Kod nekih konstruktora, moguće je čak uneti i string koji će biti implicitno konvertovan u odgovarajući tip (treba naravno biti jako pažljiv u ovakvim slučajevima)

```
my_integer_obj = int(10)
print(my_integer_obj)
my_integer_from_string = int("10")
print(my_integer_from_string)
my_integer_from_bad_string = int("hello")
print(my_integer_from_bad_string)
```

# Modul math

<https://docs.python.org/3/library/math.html>

- Operatori predstavljeni do sada ne omogućavaju automatski kompleksnije matematičke operacije (korenovanje, apsolutnu vrednost, zaokruživanje, dobavljanje sinusa i kosinusa)
- Ovaj modul takođe sadrži polja sa vredostima PI i Eulerov broj

```
import math
print(math.ceil(0.5))
print(math.floor(0.5))
print(math.sin(0.5))
print(math.cos(0.5))
print(math.pow(2,3))

print(math.pi)
print(math.e)
```

# Vežba (oopp-ex03 battlefield)

- Potrebno je kreirati jednostavnu igru u kojoj igrač bira poziciju na zamišljenoj tabli veličine 10x10 tačaka
- Nakon igračevog odabira, poziciju bira računar
- Nakon što su pozicije odabrane, prikazuje se slika sa pozicijama igrača (samo konzolna skica), i to tako što se okvir predstavlja proizvoljnim karakterom, igrač karakterom U, a kompjuter karakterom C, kao na slici:

[illegible]

# Vežba

Nakon iscrtavanja matrice, igraču se nudi mogućnost da odabere ugao. Nakon unosa ugla, prikazuje se putanja projektila (karakterom \*) kao na slici desno.

Ako se putanja ne pređe preko tačke na kojoj se nalazi slovo C, smatra se da je igrač promašio, i čeka se da računar izvrši „napad“

Računar zatim po slučajnom izboru bira ugao i ako pogodi igrača (poziciju slova U), igra se smatra završenom. U suprotnom, ponavlja se kompletna procedura biranja ugla, sve dok igrač ili računar ne pogode jedan drugog

Ukoliko dođe do pogotka, igra se završava i ispisuje se rezultat na izlazu, a zatim se ponavlja kompletna procedura (od odabira pozicija), kao na slici desno

[illegible]

```

*****
You HIT computer
*****
Total shots      6
*****
Another game (y/n)?

```

# Vežba (pomoć)

---

- Da bi od ugla i brzine dobili x i y poziciju možemo da upotrebimo sledeću formulu:  
     $x = \text{brzina} * \cos(\text{ugao});$   
     $y = \text{brzina} * \sin(\text{ugao});$

# Rad sa regularnim izrazima

<https://docs.python.org/3/library/re.html>

---

- Regularni izrazi predstavljaju način za definisanje seta karaktera kojima se formira takozvani šablon
- Za kreiranje regularnih izraza koristi se posebna sintaksa
- Regularne izraze možemo sresti/upotrebljavati na različitim mestima:
  - Veoma često u validaciji podataka
    - Validacija email adrese
    - Validacija šablona unetog podatka (na primer broj telefona ili datum)
    - ...
  - Prilikom izolovanja ključnih podataka iz nekog većeg podatka (Stringa)
    - Preuzimanje host-a iz url-a
    - Preuzimanje naslova iz teksta
    - Menjanje jedne reči u tekstu drugom rečju
    - ...

# Rad sa regularnim izrazima

<https://docs.python.org/3/library/re.html>

(oopp-ex03 simpleregex.py)

- Regularni izrazi predstavljaju, sami po sebi, jedan mali programski jezik. Ovaj jezik obično je implementiran u neku biblioteku jezika „domaćina“ kojim se njime upravlja. U Pythonu ova biblioteka je modul **re**. Zbog toga je za korišćenje regularnih izraza neophodno uvesti ovaj modul.



```
import re
```

# Kreiranje šeme (izraza) i provera Stringa

- Glavni učesnici u radu sa regularnim izrazima su klase **pattern** i **string**
- Pattern je šablon po kome treba testirati String. On se može generisati u hodu, ili kompajlirati
- Kreiranje instance Pattern klase obavlja se pomoću funkcije `compile`:

```
p = re.compile("bong")
```

- Kao parametar metode, prosleđuje se šema po kojoj će biti vršeno poređenje
- Nad objektom klase Pattern, možemo izvršavati različite metode za proveru ali je to najčešće jedna od dve (**search/match** i **sub**)

```
print("String contains word: " + result.group())  
print(re.sub(p,"HELLO","bing bang bong bung"))
```



# Šabloni regularnih izraza (oopp-ex03 regexpatterns)

---

- Prilikom krieranja šablona regularnih izraza, koristi se posebna notacija kojom se označavaju eventualne specifične situacije u tekstu za koje želimo da testiramo tekst.
- Ove situacije ne podrazumevaju fiksne vrednosti u stringu, već različite opsege vrednosti, sekvence karaktera, serije karaktera koje su približne očekivanim (fuzzy) i slično

# Opsezi

- Opseg podrazumeva poklapanje za bilo koji iz serije slučajeva. Serija slučajeva označava se uglastim zagradama:
  - **[abc]** – traže se karakteri **a**, **b** ili **c**
- Ako želimo opseg (od do) koristimo crticu (minus)
  - **[d-h]** – traže se inkluzivno karakteri od d do h (dakle d,e,f,g i h)
  - **[a-ce-g]** – traže se karakteri od a do c i od e do f
  - **[0-9]** – traže se brojevi od 0 do 9

```
print("Match if string contains numbers from 0 to 9")
p = re.compile("[0-9]")
print(p.search("I don't have any numbers"))
```

# Predefinisani karakteri u regularnim izrazima

- Za neke popularnije šablone, postoje specijalni karakteri

| Konstrukcija    | Zamena                       | Značenje                       |
|-----------------|------------------------------|--------------------------------|
| <code>\d</code> | <code>[0-9]</code>           | bilo koja cifra                |
| <code>\D</code> | <code>[^0-9]</code>          | sve osim cifre                 |
| <code>\s</code> | <code>[ \t\n\r\x0B\f]</code> | prazan karakter                |
| <code>\S</code> | <code>[^\s]</code>           | sve osim praznog karaktera     |
| <code>\w</code> | <code>[a-zA-Z_0-9]</code>    | karakter reči: slovo ili cifra |
| <code>\W</code> | <code>[^\w]</code>           | sve osim slova i cifara        |

# Vežba (oopp-ex03 regexsimplesearch.py)

---

- Treba kreirati jednu listu stringova koja sadrži rečenice

```
text = "A tank is an armoured fighting vehicle designed for front-line combat.  
Tanks have heavy firepower, strong armour, and good battlefield manoeuvrability  
provided by tracks and a powerful engine; usually their main armament is mounted  
in a turret. They are a mainstay of modern 20th and 21st century ground forces  
and a key part of combined arms combat."
```

- Potrebno je kreirati program koji uzima reč od korisnika (ili koristi hard kodiranu reč) i prikazuje rečenice u kojima se reč pojavljuje (korišćenjem regularnih izraza)

# Meta karakteri

---

- U okviru samog izraza, mogu se pojaviti karakteri koji imaju posebno značenje za evaluaciju izraza. Ovi karakteri nazivaju se metakarakter
- Osim dva koja smo već videli (uglaste zagrade i minus), postoje i drugi metakarakter `<([\^-= $!|])? * + . >`,

# Specijalni karakteri

- `.` - Jedno pojavljivanje bilo kog karaktera na mestu na kome se tačka nalazi (hel.o = hello, helao)
- `*` - Ni jedno ili više pojavljivanja prethodnog karaktera (hel\*o = helo, helllllllo)
- `+` - Jedno ili više pojavljivanja prethodnog karaktera (hel+o = hello, helllllllo (ali ne i helo))
- `?` - Karakter se može pojaviti jednom, ili ni jednom (hel?o = hello, helo)
- `{}` - Prvi parametar označava minimalan broj pojavljivanja, a drugi maksimalan (hell{2,3}o = helllo, hellllo (ali ne i hello, niti helllllo))
- `\` - Oznaka se može koristiti za ukidanje funkcionalnosti specijalnog (meta) karaktera. Specijalni karakter iza oznake backslash, biće tretiran kao običan karakter (hel\.o = hel.o (ali ne i hello, helao i slično))
- `^` - Označava početak stringa (^hello = hello (ali ne i world hello))
- `$` - Označava kraj stringa (hello\$ = world hello (ali ne i hello world))

| Kvantifikator | Značenje                               |
|---------------|--|
| X?            | jednom ili nijednom                    |
| X*            | nijednom ili više puta                 |
| X+            | jednom ili više puta                   |
| X{n}          | tačno n puta                           |
| X{n, }        | najmanje n puta                        |
| X{n, m}       | najmanje n puta, ali ne više od m puta |

# Vežba – validacija broja telefona

(oopp-ex03 regexphonenovalidate.py)

---

- Potrebno je izvršiti validaciju broja telefona po sledećem šablonu: `###/###-####`, tako da na primer, sledeći string bude validan: 123/456-7890

# Zadatak

---

- Pokušajte samostalno da kreirate i primenite šablon za validaciju email adrese