Univerzitet Crne Gore
Elektrotehnički fakultet

Prof. dr Vesna Popović-Bugarin

# Uvod u Experta (PyKnow)

- Deseti termin -

# Paradigma programiranja

- **Na pravilima zasnovano programiranje** – Rule-based language.
- Inspirisan CLIPS programskim jezikom.
- CLIPS je programski jezik, dok je experta biblioteka u okviru Python programskog jezika.
- Ideja je da se zadrži paradigma programiranja kao u CLIPS-u, ali da se poveže sa Python-om.
- Ipak, postoje bitne razlike i mora se znati dio sintakse Python-a.
- CLIPS nije zahtijevao poznavanje C/C++ programskog jezika

# Komponente experta-a

- experta pokriva osnovne komponente ekspertnog sistema:
  - Liste činjenica
  - Bazu znanja
  - Mašinu za zaključivanje – Rete algoritam
    - Pravilima pridružuje odgovarajuće činjenice, bira pravila koja su zadovoljena i koja će biti ‘okinuta’ i izvršava akcije definisane aktiviranim pravilom.
    - Prilikom odlučivanja koje će pravilo biti prvo aktivirano, prioritet imaju pravila koja zavise od većeg broja činjenica, ili od više osobina neke činjenice.
    - U jednoj iteraciji se izvršava jedno pravilo
    - Činjenica je u experta biblioteci zapravo klasa sa određenim osobinama, izvodi se iz nadklase Fact koja je ugrađena klasa experta-a

# Crash kurs Python-a

- Nema uglastih zagrada za odvajanje blokova koda, Python koristi uvlačenje (engl. identation)

```
for i in [1, 2, 3, 4, 5]:#lista
    print(i) # prva linija "for i" blok
    for j in [1, 2, 3, 4, 5]:
        print(j) # prva linija "for j" blok
        print(i + j) # zadnja linija "for j" blok
    print("\n")
    print(i) # zadnja linija "for i" blok
print("done looping")
```

- Razmaci unutar zagrada se ignorišu

```
In [17]: ignorisi = (1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 )

In [18]: ignorisi
Out[18]: 45
```

# Crash kurs Python-a

```
range (start, stop[, step])
```

```
for x in range(10):
    print x, "is less than 10"
```

- Start i step su opcioni, indeks prvog elementa niza je 0

```
In [39]: x = range(10)
         print(x[1])
         x = range(2,10)
         print(x[1])
         x = range(2,10,2)
         print(x[1])
```

```
1
3
4
```

```
prvaDva = (x[:2])
for i in prvaDva:
    print(i)
```

```
2
4
```

```
In [47]: posljednjaDva = x[2:]
         odTrecegDoPetog = x[3:5]
         bezprvogiposljednjeg = x[1:-1]
         DoPretposljednjeg = x[:-1]
```

```
In [48]: for i in DoPretposljednjeg:
             print(i)
```

```
2
4
6
```

# Crash kurs Python-a

- Određene funkcije Python-a nisu podrazumijevano učitane. Ovo mogu biti funkcije koje su dio jezika ili funkcije koje sami preuzmamo kroz biblioteke, kao npr. experta biblioteku. Da bi se ove funkcije koristile, moraju se uključiti moduli koje ih sadrže.

```
from experta import *
#uključuje kompletan modul
from experta import Fact
#uključuje samo Fact
```

- Ako već u kodu imamo ime modula, može se uvesti alijas

```
import re as regex
import matplotlib.pyplot as plt
```

# Crash kurs Python-a

- Rezultat dijeljena dva cijela broja je podrazumijevano cijeli broj u starijim verzijama, u novijim nije

```
In [21]: 5/2
Out[21]: 2.5

In [22]: 5//2
Out[22]: 2
```

Funkcije se mogu pridruživati promjenljivim i prosleđivati funkcijama kao bilo koji drugi argument drugim

```
In [23]: def dupliraj(x):
             return 2*x
         def primijeniNaDva(f):
             """Pozvaće funkciju f za argument 2"""
             return f(2)
         duplo = dupliraj
         x = primijeniNaDva(dupliraj)

In [24]: print(x)
4
```

# Crash kurs Python-a

- Mogu se jednostavno kreirati bezimene funkcije

```
In [27]: y = primijeniNaDva(lambda x : x + 4)
         print(y)

         6
```

- Funkcije mogu imati argumente sa podrazumijevanim vrijednostima. Ovi argumenti se moraju specificirati samo kada želimo drugačiju vrijednost od podrazumijevane

```
In [29]: def MojeStampanje(poruka = "želite da učite Python?"):
             print(poruka)
         MojeStampanje()
         MojeStampanje("Sada houću ovu poruku")

         želite da učite Python?
         Sada houću ovu poruku
```

```
def oduzmi(a=0,b=0):
  return a-b
oduzmi(10,5) # 5
oduzmi(10,-5) # 15
oduzmi(5) # 5
oduzmi(b=5) # -5
```

# Crash kurs Python-a

- Stringovi se mogu zadavati sa jednostrukim ili dvostrukim navodnicima. Trostruki služe za kreiranje stingova u više redova

```
In [31]: Apostrof = 'Neki string'
         Navodnici = "Neki string"
         StingUViseRedova = """ Za ovo
         mi trebaju trostruki navodnici"""
```

- Dictionary – još jedan fundamentalni tip podatka. Čini ga skup parova key:value. Indeksirani su jedinstvenim ključevima - key

```
In [60]: ocjene = {"Marko" : 10,"Petar" : 5,"Ilija":7}
         if "Ilija" in ocjene:
             print(ocjene["Ilija"])
```

```
7
```

Pretražuje po ključu

```
In [62]: ocjene["Sanja"] = 5
         len(ocjene)
```

```
Out[62]: 4
```

Dodali smo novi par

# Crash kurs Python-a

- Kada su ključevi Dictionary-ja stringovi, nekada je jednostavnije koristiti dict() konstruktor – biće kasnije više riječi o konstruktorima

```
In [28]: ocjene = dict(Marko = 10,Petar = 5,Ilija=7)

In [31]: list(ocjene) #izlistava keys

Out[31]: ['Marko', 'Petar', 'Ilija']
```

- List izlistava ključeve korišćene u Dictionary-ju.

# Crash kurs Python-a

- Python podržava rad sa klasama. Podaci članovi se ne moraju navoditi. Podrazumijevano su javni i mogu se kasnije imenovati i dodijeliti im se vrijednosti

```
In [64]: class NekaKlasa:
             pass
         a = NekaKlasa()
         a.ime = "Vesna"
         a.prezime = "Popovic"

In [65]: print(a.ime)

         Vesna
```

- Inicijalizacija klasom koristi notaciju funkcija – ().
- Podacima članovima i f-jama članicama se pristupa preko operatora . (dot).

# Crash kurs Python-a

- Uobičajenije je da se imenovanje podataka članova i inicijalizacija vrši konstruktorom

```
In [68]: class KlasaSaKonstruktorom:
             def __init__(self,ime,prezime):
                 self.ime = ime
                 self.prezime = prezime
```

```
In [69]: a = KlasaSaKonstruktorom("Vesna","Popovic")#self se ne prosledjuje eksplicitno
```

- Možemo dodati osobine koje nijesu pomenute u init metodi

```
In [76]: print(a.ime)
         a.ocjena = 10
         print(a.ocjena)
```

```
Vesna
10
```

# Crash kurs Python-a

- I u Pythonu postoji nasleđivanje

```
In [77]: class Izvedena(KlasaSaKonstruktorom):
             pass

In [79]: b = Izvedena("Vesna","Popovic")

In [80]: print(b.ime)
         Vesna
```

# Facts - činjenice

- Činjenica predstavlja “komad” informacije
- Fact je ugrađena klasa, podklasa klase dict(). Da bi se koristila treba biti instalirana PyKnow biblioteka. To činimo kucanjem `$ pip install pyknow (experta` od *Python 3.0)* u komandnoj liniji Python-a. Sada se *Fact može koristiti.*

```
In [82]: F = Fact(a = 2,b = 3)
         F["a"]

Out[82]: 2
```

- Za razliku od dict(), može se kreirati činjenica – Fact bez ključeva, već samo sa vrijednostima. Fact će automatski generisati numeričke ključeve za unijete vrijednosti. Ovo su uređene činjenice, bitan je redoslijed – jer je on ključ.

```
In [83]: f = Fact('x','y','z')
         f[0]

Out[83]: 'x'
```

# Facts - činjenice

- Mogu se miješati autonumeričke vrijednosti sa vrijednostima zadatim preko ključeva Key, samo je bitno da autonumeričke budu deklarisane prve

```
In [87]: f = Fact('x','y','z',a = 2, b = 3)
         f[2]

Out[87]: 'z'

In [88]: f['a']

Out[88]: 2
```

- Vrijednostima elemenata zadatih preko ključa se mora i pristupati tako, dakle mora f[‘a’], ne može f[3]

# Facts - činjenice

- Kada želimo napraviti neku činjenicu sa konkretnim osobinama i/ili vrijednostima, ili joj zadati dodatne funkcionalnosti, mi je zapravo izvodimo iz klase Facts

```
In [90]: class ZamisliNesto(Fact):
             pass
         f1 = ZamisliNesto("Zivotinja")
         f2 = ZamisliNesto("Biljka")
```

```
In [91]: print(f1[0])
```

```
Zivotinja
```

```
In [92]: f3 = ZamisliNesto("Zivotinja",vrsta = "Mačka")
         print(f3["vrsta"])
```

```
Mačka
```

# KnowledgeEngine

- Glavna klasa u kojoj se cijela magija dešava.
- Konkretan ES pišemo kao podklasu ove klase
- Nakon toga koristimo ključnu riječ @*Rule* da bismo pisali pravila za rješavanje konkretnog problema
- Kada imamo sva željena pravila, možemo inicijalizovati ovu klasu, popunjavati je činjenicama i na kraju pokrenuti ES
- Pravila se sastoje od dvije komponente LHS (left-hand-side) i RHS (right-hand-side).
- LHS opisuje (koristeći šablone - klase) uslove pod kojim pravilo treba biti izvršeno ili „okinuto“. Ovi šabloni opisuju karakteristike činjenica koje zadovoljavaju konkretno pravilo.

# Pravila

- Da bi neko pravilo bilo zadovoljeno, sva ograničenja šablona moraju biti ispunjena
- RHS je skup akcija koje se trebaju izvršiti kada se izvrši pravilo. RHS je funkcija članica, koja kao obavezan argument ima objekat klase za koji se poziva – obično označen sa *self*.
- Pravilo se može zamisliti kao

IF Postoje činjenice koje zadovoljavaju zadate šablone – LHS

THEN

Uradi nešto – RHS

sintaksa

```
@Rule(<sablon1>,<sablon2>, …,<sablon n>) #LHS
        def ime(self): #RHS
            pass
```

# KnowledgeEngine

```
class MojaCinjenica(Fact):
    pass
class Ilustracija(KnowledgeEngine):
    @Rule(MojaCinjenica()) #Ovo je LHS
    def uparujSaBiloKojomCinjenicom(self):
        """Ovo pravilo se uparuje sa bilo kojom instancom MojaCinjenica"""
        # Ovo je RHS
        print("ImasNekuCinjenicu")
    @Rule(MojaCinjenica('zivotinja',vrsta = 'macka'))
    def MackaJe(self):
        print("Macka je")
```

Drugo pravilo zadovoljavaju samo instance - objekti *MojaCinjenica* koje imaju :

`f[0] == 'zivotinja'`

`f['vrsta'] == 'macka'`

# Kako funkcioniše pravilo?

- LHS – Pravilo može zavisiti od više šablona (činjenica). Svaki šablon se sastoji od nula, jednog ili više uslova koji se pokušavaju spojiti sa činjenicama u listi.
- **Pravilo se aktivira** ako svi njegovi uslovi odgovaraju nekim činjenicama iz liste – postoje činjenice u listi čiji parametri zadovoljavaju LHS pravila.
- Ako pravilo nema nijedan uslov-šablon experta će automatski dodati **(InitialFact())** kao uslov pravila. To pravilo će se aktivirati uvijek nakon naredbe **reset** kojom se generiše činjenica *InitialFact*.
- Aktivirana pravila se ubacuju u **agendu.**
- Agenda je lista pravila čiji su uslovi zadovoljeni, a koja još uvek nisu izvršena. U agendi može biti nula ili više pravila.

# Kako funkcioniše pravilo?

- Pravila u agendi *experta* uređuje u opadajući poredak po prioritetu i prvo se izvršava pravilo sa najvišim prioritetom. Ukoliko se ne dodijeli prioritet, automatski je nula, i prvo se izvršava zadnje aktivirano pravilo.
- Ukoliko postoje pravila sa većim brojem uslova koje moraju zadovoljiti, ona imaju prioritet
- **Izvršeno pravilo se uklanja iz agende i neće biti ponovo aktivirano istim skupom činjenica iz liste kojim je bilo aktivirano prvi put!!!**

# Dodavanje i brisanje činjenica

- Sve činjenice u experta se nalaze u listi činjenica (*FactList*) za konkretnu podklasu *KnowledgeEngine()*
- Činjenice koje predstavljaju informacije se mogu dodavati u listu, i uklanjati iz liste.
- `declare()` – dodaje činjenicu u listu
- `retract()` – briše činjenicu iz liste
- `facts` – izlistava činjenice – statička promjenljiva, nema zagrade
- Sve su ovo članice klase *KnowledgeEngine* i ima ih smisla koristiti u njenim podklasama.

# Dodavanje i brisanje činjenica

```
In [34]: engine = KnowledgeEngine()
         engine.reset()
         engine.declare(Fact(Ocjena = 10))

Out[34]: Fact(Ocjena=10)

In [36]: engine.facts

Out[36]: FactList([(0, InitialFact()), (1, Fact(Ocjena=10))])
```


- Lista činjenica je skup parova (indeks, činjenica).
- Preko indeksa se pristupa činjenicama kako bi se brisale, modifikovale ili duplirale.
- `InitialFacts()` se formira sa `engine.reset()` i uvijek ima indeks 0

# Prikazivanje liste činjenica

```
engine.reset()
d = engine.declare(Fact(Ocjena = 10))
```

```
In [36]: engine.facts
Out[36]: FactList([(0, InitialFact()), (1, Fact(Ocjena=10))])

In [38]: d
Out[38]: Fact(Ocjena=10)

In [39]: engine.declare(Fact(Ocjena = 15))
Out[39]: Fact(Ocjena=15)

In [40]: engine.facts
Out[40]: FactList([(0, InitialFact()), (1, Fact(Ocjena=10)), (2, Fact(Ocjena=15))])

In [41]: engine.retract(1)

In [42]: engine.facts
Out[42]: FactList([(0, InitialFact()), (2, Fact(Ocjena=15))])
```

# Prikazivanje liste činjenica

```
In [50]: engine.retract(2)

In [51]: engine.facts
Out[51]: FactList([(0, InitialFact())])
```

```
[24] engine.declare(Fact(Ocjena = 7))

     Fact(Ocjena=7)
```

```
engine.facts

FactList([(0, InitialFact()), (3, Fact(Ocjena=7))])
```

Indeks novododate činjenice se povećava kao da su sve prethodne činjenice i dalje u memoriji, nema ponavljanja indeksa. Činjenica se može brisati i modifikovati i navođenjem njenog indeksa iz *FactList*

# Uklanjanje i modifikacija činjenica

- *modify*() uklanja originalnu činjenicu i dodaje novu sa modifikovanim vrijednostima specificiranih slotova-promjenljivih.
- *duplicate*() ne uklanja originalnu činjenicu, već samo dodaje duplikat sa navedenom izmjenom.
- **Modify i duplicate se ne mogu upotrebljavati sa uređenim činjenjicama.**

```
In [56]: engine.modify(engine.facts[3],ocjena = 9,boja = 'crvena')
         engine.facts

Out[56]: FactList([(0, InitialFact()), (4, Fact(ocjena=9, boja='crvena'))])

In [57]: engine.duplicate(engine.facts[4],boja = 'zelena')

Out[57]: Fact(ocjena=9, boja='zelena')

In [58]: engine.facts

Out[58]: FactList([(0, InitialFact()),
                   (4, Fact(ocjena=9, boja='crvena')),
                   (5, Fact(ocjena=9, boja='zelena'))])
```

- Činjenica sa indeksom 3 se nakon modify izbacuje iz memorije i modifikovana ubacuje pod novim indeksom - 4

# Konstrukcija deffacts

- Koristi se za automatsko istovremeno dodavanje više činjenica.
- Pogodna je za unošenje činjenica koje su poznate i neophodne prije izvršenja programa i za testiranje programa.
- Sintaksa

```
@DefFacts()
def imeListe(self):
  yield(cinjenica(osobine))
```

**Prije deffacts mora biti definisana klasa za činjenice koje se njom unose**

# Komanda reset

- Naredba `(reset)` uklanja sve činjenice iz liste i u listu ubacuje činjenicu **(**`InitialFact()`**)** i sve činjenice zadate **deffacts** komandom.
- **Tek nakon komande reset, unose se činjenice zadate deffacts komandom!**
- Naredba reset se najčešće piše na početku programa kako bi se unijela početno znanje – BZ.
- InitialFact() uvijek ima indeks 0. Može joj se pristupiti preko engine.facts[0].
- Često se koristi za startovanje izvršavanja programa.

# Komanda reset

In [69]:
```
class primjer(KnowledgeEngine):
    @DefFacts()
    def NekeCinjenice(self):
        yield Fact(ime = 'Vesna',Prezime = 'popovic')
        yield Fact(boja = 'crvena')
        yield Fact('pokreni se')
```

In [70]:
```
engine = primjer()
engine.reset()
engine.facts
```

Out[70]:
```
FactList([(0, InitialFact()),
          (1, Fact(ime='Vesna', Prezime='popovic')),
          (2, Fact(boja='crvena')),
          (3, Fact('pokreni se'))])
```

In [71]:
```
engine.facts[0]
```

Out[71]:
```
InitialFact()
```

# Izvršavanje pravila

- Sintaksa: `engine.run()`
- Pravila će se izvršavati sve dok se ne isprazni agenda.

# Primjer

```
[4]  from experta import *
     class ZamisliNesto(Fact):
       pass

     class Ilustracija(KnowledgeEngine):
       @DefFacts()
       def NekeCinjenice(self):
         yield(ZamisliNesto())
         yield(ZamisliNesto('Zivotinja',vrsta='Macka'))
       @Rule(ZamisliNesto())
       def ImasNesto(self):
         print("imas neku cinjenicu ZamisliNesto")
       @Rule(ZamisliNesto('Zivotinja',vrsta = 'Macka'))
       def ImasMacku(self):
         print("Imas macku")
```

# Primjer ...

```
engine = Ilustracija()
```

```
[6] engine.reset()
```

Unosi InitialFact() i DefFacts()

```
[7] engine.run()
```

```
imas neku cinjenicu ZamisliNesto
Imas macku
imas neku cinjenicu ZamisliNesto
```

```
[8] engine.facts
```

```
FactList([(0, InitialFact()),
          (1, ZamisliNesto()),
          (2, ZamisliNesto('Zivotinja', vrsta='Macka'))])
```

```
[9] engine.run()
```

Iste činjenice ne mogu više puta pokrenuti isto pravilo

# Primjer ...

```
engine = Ilustracija()
```

```
[6]  engine.reset()
```

```
[7]  engine.run()
```

```
imas neku cinjenicu ZamisliNesto
Imas macku
imas neku cinjenicu ZamisliNesto
```

```
[8]  engine.facts
```

```
FactList([(0, InitialFact()),
          (1, ZamisliNesto()),
          (2, ZamisliNesto('Zivotinja', vrsta='Macka'))])
```

```
[9]  engine.run()
```

Iste činjenice ne mogu više puta pokrenuti isto pravilo

# Primjer ...

```
[13] engine.declare(ZamisliNesto('Biljka'))
```

```
ZamisliNesto('Biljka')
```

```
[14] engine.facts
```

```
FactList([(0, InitialFact()),
          (1, ZamisliNesto()),
          (2, ZamisliNesto('Zivotinja', vrsta='Macka')),
          (3, ZamisliNesto('Biljka'))])
```

```
[15] engine.run()
```

```
imas neku cinjenicu ZamisliNesto
```

Samo novododata činjenica pokreće pravilo – i to prvo pravilo