Математички факултет

Универзитет у Београду

МАСТЕР РАД

Објектно оријентисана парадигма у програмском језику Пајтон

Ментор:
Доц. др Мирослав Марић

Кандидат:
Маја Михајловић

Београд, август 2014.

Садржај

1. Увод

Овај мастер рад представља део курса за програмски језик Пајтон, замишљен за почетнике програмере са жељом да приближи програмирање и начин размишљања при програмирању. Програмски језик Пајтон изабран је из више разлога, пре свега, Пајтон је језик близак говорном језику и зато га је лакше усвојити него остале програмске језике. Синтаксно није тежак и много се више ослања на размаке при креирању блокова, него на карактере било које врсте, што доприноси лакој читљивости и јасном одвајању блокова и подблокова, што, иако је препоручена програмерска пракса, није неопходно и често се игнорише. Користећи Пајтон ученици ће научити да поштују то правило, јер је неопходно у овом програмском језику. Синтакса је уједно и предност и мана овог програма. Када се ради о почетницима, синтаксу је могуће боље научити на једноставнијем програмском језику (што Пајтон и јесте) него на компликованијем, међутим, уколико програм написан у Пајтону треба превести на други програмски језик то може бити проблематично управо због особености синтаксе Пајтона [1].

Још једна од предности, али такође на неки начин и мана Пајтона јесте недостатак одређивања типа променљивих. За почетника је лакше да не мора експлицитно да одређује тип променљивих, а такође је лакше написати једну функцију која ће радити независно од типа променљивих које јој се шаљу. С друге стране, при преласку на друге програмске језике можда ће на почетку бити проблема око декларисања, али се то брзо да превазићи.

Приступ променљивама у Пајтону није ограничен, као ни креирање нових у сваком тренутку, што може умногоме помоћи новом програмеру, јер нема потребе да размишља о томе у ком тренутку може, а у ком не може да приступи променљивој и њеној вредности.

Када се конкретно ради о објектно оријентисаном програмирању и класама, кроз рад ће бити приказано како се лако и логично креирају класе, њихове методе као и подкласе. Пајтон је добар као основа за рад управо због тога што покрива основе на једноставан начин, а даје могућности и за компликованије програме, добро је подржан на свим платформама и има широку примену.

1.1. Историјат

Творац Пајтон програмског језика је Гидо вон Росум, холандски програмер, који га је представио 1991. године [2]. Са жељом да свима приближи програмирање, радио је на томе да синтаксу програмирања у овом програмском језику учини једноставном, разумљивом као и логичном на језичком нивоу. Свој пројекат је назвао Пајтон, јер је обожавалац Летећег циркуса Монтија Пајтона. Основна идеја програмирања у Пајтону је да се све може одрадити на један начин и то најлогичнији. Данас иза најновијих верзија овог програмског језика стоји и мноштво волонтера који су помагали у његовом развоју, чинећи га све бољим [3]. Пајтон је лако доступан и бесплатан, што је и била замисао његовог творца.

1.2. Основни концепти и карактеристике

Као што је већ речено, замисао при креирању овог језика била је да буде једноставан, што он и јесте. Лак је за учење, основни концепти се брзо савладавају и по многима је идеалан као први програмски језик управо због тога што је близак енглеском језику [4].

Овај програмски језик је објектно оријентисан, тј. све што се ради у њему представља објекат, све постојеће структуре, али и све структуре које сам корисник направи. Објекат комбинује већ постојеће структуре у скуп који представља нову расположиву структуру. Циљ оваквог начина програмирања је лак рад са разноврсним типовима података и њихова организација у логичне целине.

Пајтон је интерпретерски језик, што значи да се при превођењу на машински језик преводи линија по линија кода и тако се и извршава. Он се може користити на свим оперативним системима, располаже богатом базом структура података и функција и његова примена је широко распрострањена [1].

1.3. Концепт објектно оријентисаног програмирања

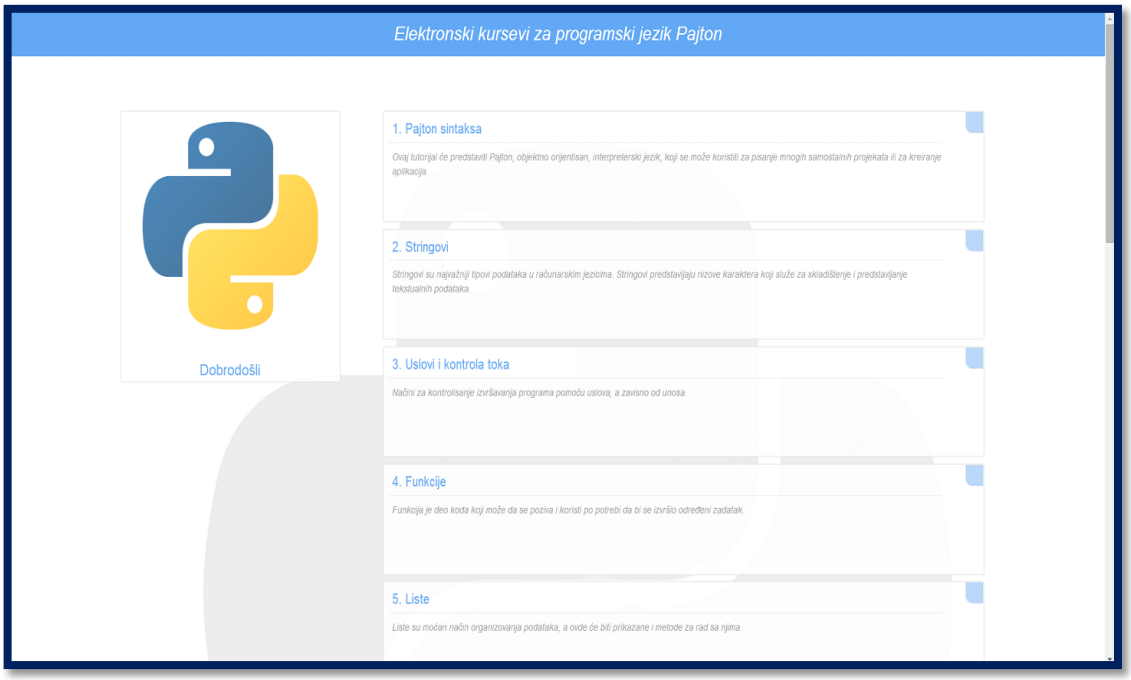
Објектно оријентисано програмирање је суштински начин организовања програма тако се упакују подаци и функције које раде са њима унутар нечега што се назива објектом. Овакав начин програмирања, наравно, није обавезан, али умногоме помаже код различитих проблема. Најбитнији појмови објектно оријентисаног програмирања су објекти и класе. Објекти представљају све податке са којима се ради, па и оне најједноставније и уобичајене типове као што су бројеви или ниске. Помоћу класе се прави објекат одређеног типа унутар кога се налазе једноставнији објекти (називани елементи, атрибути или поља објекта). Унутар класе се налазе функције које се могу користити за објекте истог типа и такве функције се називају методе [5].

2. Сајт Електронски курсеви за програмски језик Пајтон

Сајт електронски курсеви за програмски језик Пајтон налази се на следећој адреси:

http://edusoft.matf.bg.ac.rs/python/index.html

Почетна страна сајта се може видети на слици 1.

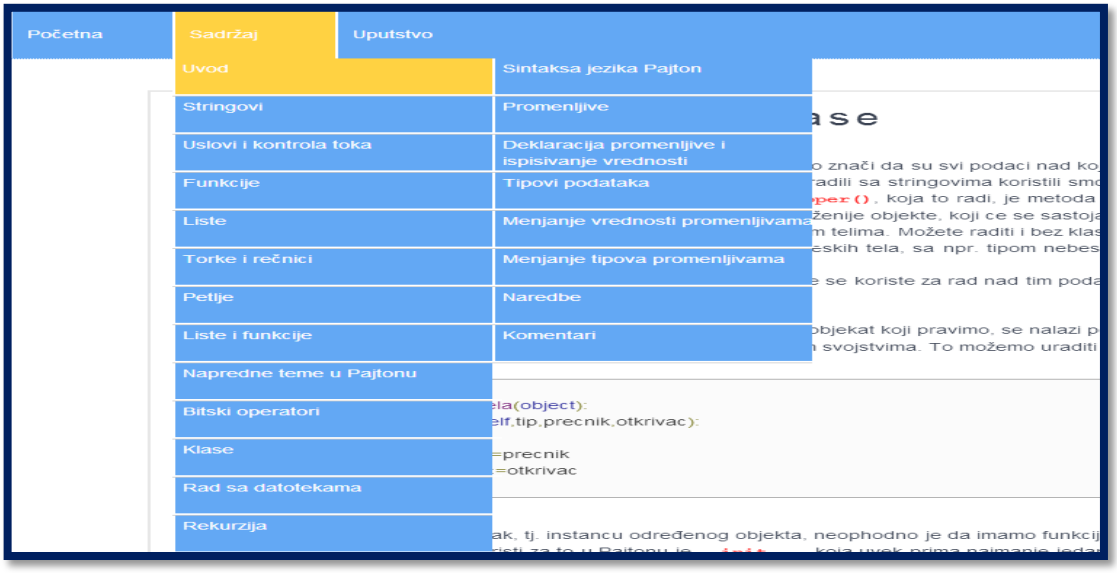


Слика 1. Почетна страна сајта

На почетној страни се налазе имена тринаест лекција, као и кратак опис. Кликом на наслов лекција долази се до почетне стране сваке од њих. Лекције које се тренутно налазе на сајту су:

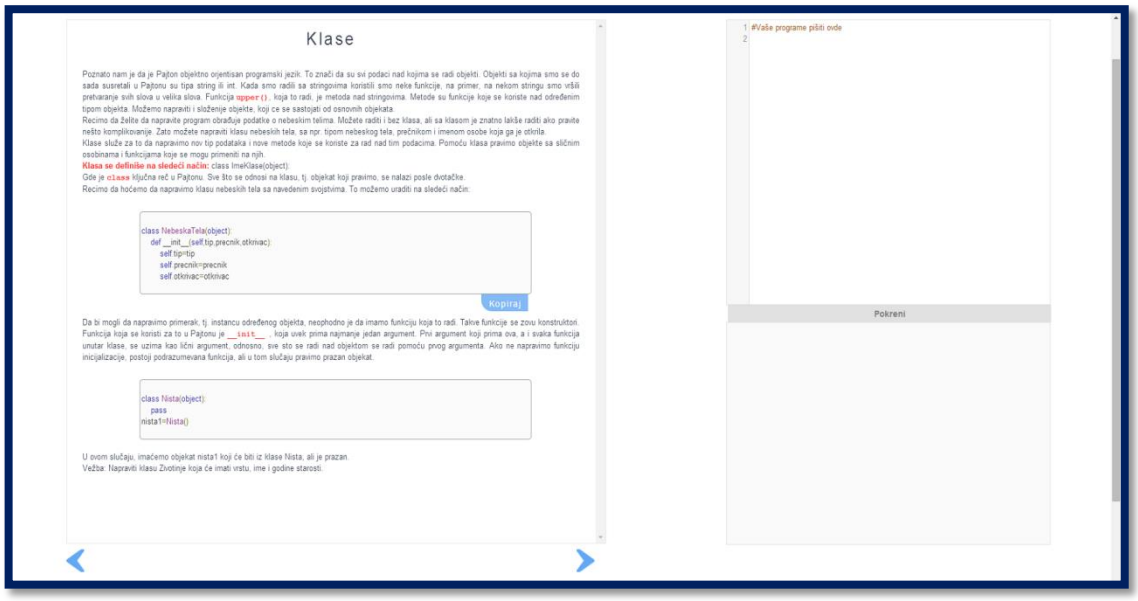
1. Синтакса Пајтона
2. Стрингови
3. Услови и контрола тока
4. Функције
5. Листе
6. Торке и речници
7. Петље
8. Листе и функције
9. Напредне теме у Пајтону
10. Битски оператори
11. Класе
12. Рад са датотекама
13. Рекурзија

Оно што је заједничко за сваку страну јесу заглавље стране, дно стране, место за лекцију и конзола са исписом. У заглављу стране (слика 2.) налази се линк ка почетној страни, садржај у оквиру кога се може приступити свакој лекцији и подлекцији, као и упутство у којем се објашњава како направити локалну верзију Пајтона и која верзија је коришћена на сајту тј. верзија 3.3.5.



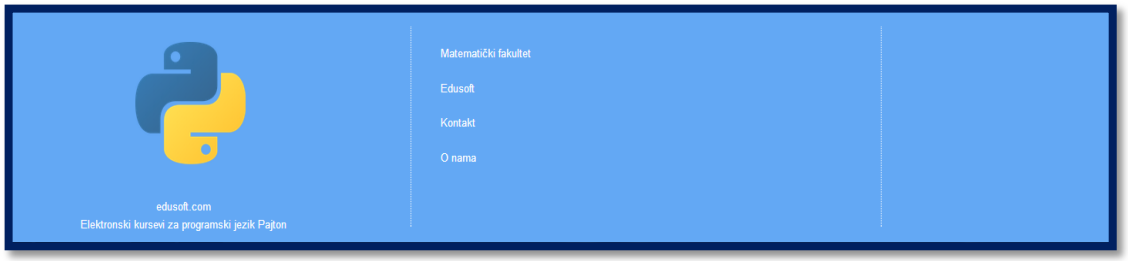
Слика 2. Заглавље стране

Главни део стране за лекције обухвата место за текст лекција са леве стране и конзолу са десне стране. У оквиру лекција се налазе и кодови који се на једноставан начин могу ископирати у конзолу, кликом на дугме које се налази испод кода (копирање је могуће уколико кодови дају неки испис). Конзола је такође доступна за писање самосталних програма који се извршавају притиском на дугме Pokreni. Испод сваке лекције налазе се стрелице које воде ка претходној и ка наредној лекцији (Слика 3.).



Слика 3. Главни део стране

У дну стране налази се део који садржи лого Пајтона и линкове ка основним странама Математичког факултета, едусофта, као и линкове ка контакту и страни О nama које ће накнадно бити урађене (Слика 4.).



Слика 4. Дно стране

У тренутку када ученик дође до лекције Класе, рачунајући да је прешао све претходне лекције, већ је упознат са основама програмског језика Пајтон, са синтаксом, начином рада са променљивама, функцијама, петљама итд. На крају сваке лекције, у

Класа делу, ученици имају задатак да напишу део кода који се односи на управо пређени део. Задатак је ту да би навео ученике да испробају писање у конзоли и исписивање кода, али за сада није подржано проверавање исправности. Лекције су подељене логички, мада некад и на више страна да би се ученицима олакшао рад и да не би били преоптерећени превеликим бројем нових информација.

3. Основно о класама

Објектно оријентисан концепт програмирања је заснован на томе да су сви типови података са којима се ради – објекти. Такође, могуће је правити нове објекте засноване на старим објектима. Ту је кључан појам класа. Класа је структура која омогућава прављење нових објеката, као и прављење метода (функција) које раде са тим објектима. Пајтон, као сваки објектно оријентисан програмски језик, омогућава креирање класа а самим тим и објеката на врло лак начин који ће бити приказан у наредним поглављима.

3.1. Креирање класа у Пајтону

Креирање нове класе у Пајтону је једноставно и постиже се на следећи начин:

```
class Ime_Klase(object):
```

Где је `class` кључна реч за креирање класе, затим је наведено име класе и унутар заграда, основни објекат – `object`. Иначе, унутар заграда се налази класа од које се наслеђују својства, али о томе нешто више у наставку.

Пре креирања нове класе, мора се знати које врсте података (атрибута) је потребно имати. Ако је нпр. потребан објекат који ће представљати студента, неопходно је да у оквиру класе која то репрезентује постоји барем име и презиме тог студента, односно одговарајуће стринг инстанце. Важно је напоменути да, иако Пајтон сам препознаје типове објекта, при креирању класе треба бити опрезан да се

одговарајуће функције могу применити на прављени објекат и да се типови постојећих података увек пажљиво одреде и запамте, иако нису експлицитно наведени у коду.

У оквиру ове лекције на курсу је направљена класа небеских тела, која као атрибуте небеског тела садржи његов тип (звезда, планета, комета итд.), ког је пречника и ко га је открио. Најједноставнији код за овакву класу може се видети у примеру:

```
class NebeskaTela(object):

    def __init__(self,tip,precnik,otkrivac):

        self.tip=tip

        self.precnik=precnik

        self.otkrivac=otkrivac
```

Може се приметити да, за разлику од других објектно оријентисаних програмских језика, у Пајтону при креирању класе се не наводе типови података који се налазе у класи на уобичајен начин (тип податка и име), већ се то ради кроз функцију `__init__`. Ова функција се користи за иницијализацију, дакле, за креирање инстанце типа `Nebeska tela`. Свака функција унутар класе увек прима најмање један аргумент, који се узима као лични аргумент инстанце, односно, све што се ради са објектом ради се помоћу овог аргумента. Најчешће је тај аргумент под називом `self`, што није неопходно али је прихваћени стандард.

Ако се не направи функција иницијализације, постоји уграђена подразумевана функција, али у том случају се прави празан објекат и непознато је од којих се типова састоји. Код функције иницијализације није неопходно да се аргументи које функција прима називају исто као и аргументи које ће објекат да садржи (`self.tip` говори да објекат има `tip` као податак док `=tip` додељује вредност, али не одређује име податка). То је могуће урадити и на другачији начин, као у примеру који следи.

```
class NebeskaTela(object):

    def __init__(self,arg1,arg2,arg3):

        self.tip=arg1
```

```
        self.precnik=arg2

        self.otkrivac=arg3
```

Опет, уобичајено је да се иницијализација прави на први начин пошто је јаснији. У електронском курсу, овај податак и податак за `self` аргумент нису експлицитно објашњени, пошто је сматрано да није неопходно за програмере почетнике јер не отвара нове могућности програмирања у Пајтону. Изглед ове лекције у оквиру курса на сајту може се видети на слици 5.

Klase

Poznato nam je da je Pajton objektno orjentisan programski jezik. To znači da su svi podaci nad kojima se radi objekti. Objekti sa kojima smo se do sada susretali u Pajtonu su tipa string ili int. Kada smo radili sa stringovima koristili smo neke funkcije, na primer, na nekom stringu smo vršili pretvaranje svih slova u velika slova. Funkcija `upper()`, koja to radi, je metoda nad stringovima. Metode su funkcije koje se koriste nad određenim tipom objekta. Možemo napraviti i složenije objekte, koji ce se sastojati od osnovnih objekata.
Recimo da želite da napravite program obrađuje podatke o nebeskim telima. Možete raditi i bez klasa, ali sa klasom je znatno lakše raditi ako pravite nešto komplikovanije. Zato možete napraviti klasu nebeskih tela, sa npr. tipom nebeskog tela, prečnikom i imenom osobe koja ga je otkrila.
Klase služe za to da napravimo nov tip podataka i nove metode koje se koriste za rad nad tim podacima. Pomoću klasa pravimo objekte sa sličnim osobinama i funkcijama koje se mogu primeniti na njih.
Klasa se definiše na sledeći način: class ImeKlase(object):
Gde je `class` ključna reč u Pajtonu. Sve što se odnosi na klasu, tj. objekat koji pravimo, se nalazi posle dvotačke.
Recimo da hoćemo da napravimo klasu nebeskih tela sa navedenim svojstvima. To možemo uraditi na sledeći način:

```
class NebeskaTela(object):
    def __init__(self,tip,precnik,otkrivac):
        self.tip=tip
        self.precnik=precnik
        self.otkrivac=otkrivac
```

Kopiraj

Da bi mogli da napravimo primerak, tj. instancu određenog objekta, neophodno je da imamo funkciju koja to radi. Takve funkcije se zovu konstruktori. Funkcija koja se koristi za to u Pajtonu je `__init__` , koja uvek prima najmanje jedan argument. Prvi argument koji prima ova, a i svaka funkcija unutar klase, se uzima kao lični argument, odnosno, sve sto se radi nad objektom se radi pomoću prvog argumenta. Ako ne napravimo funkciju inicijalizacije, postoji podrazumevana funkcija, ali u tom slučaju pravimo prazan objekat.

```
class Nista(object):
    pass
nista1=Nista()
```

U ovom slučaju, imaćemo objekat nista1 koji će biti iz klase Nista, ali je prazan.
Vežba: Napraviti klasu Zivotinje koja će imati vrstu, ime i godine starosti.

Слика 5. Прва лекција - Класе

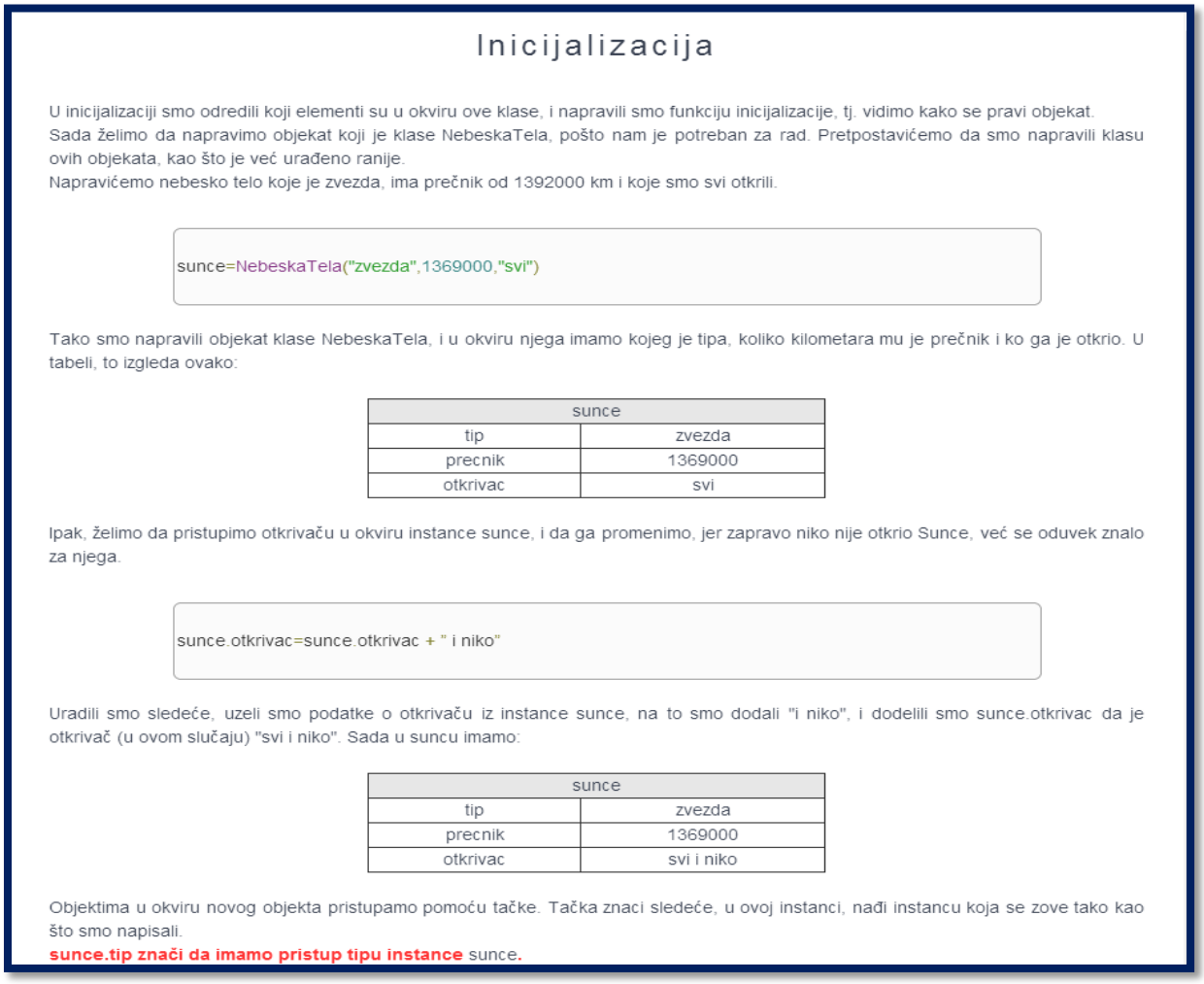
3.2. Иницијализација

Иницијализација је одвојена као нова лекција и ту се посебна пажња посветила схватању појма објекта и на који начин се подобјекти налазе у објекту. Прављење инстанце је јако једноставно и логично:

```
sunce=NebeskaTela("zvezda",1369000,"svi")
```

На овај начин направљена је инстанца `sunce` са `zvezda` као `tip`-ом небеског тела, `1369000` представља `precnik`, а као `otkrivac`-и су наведени `svi`.

Због претходног искуства аутора са објашњавањем појма објекта и класа, као и због успешности истог са табеларним приступом, ученицима је приказана мала табела са подацима конкретног објекта односно инстанце која је направљена у претходном реду, као што се може видети на слици 6.



Inicijalizacija

U inicijalizaciji smo odredili koji elementi su u okviru ove klase, i napravili smo funkciju inicijalizacije, tj. vidimo kako se pravi objekat.
Sada želimo da napravimo objekat koji je klase NebeskaTela, pošto nam je potreban za rad. Pretpostavićemo da smo napravili klasu ovih objekata, kao što je već urađeno ranije.
Napravićemo nebesko telo koje je zvezda, ima prečnik od 1392000 km i koje smo svi otkrili.

```
sunce=NebeskaTela("zvezda",1369000,"svi")
```

Tako smo napravili objekat klase NebeskaTela, i u okviru njega imamo kojeg je tipa, koliko kilometara mu je prečnik i ko ga je otkrio. U tabeli, to izgleda ovako:

sunce	
tip	zvezda
precnik	1369000
otkrivac	svi

Ipak, želimo da pristupimo otkrivaču u okviru instance sunce, i da ga promenimo, jer zapravo niko nije otkrio Sunce, već se oduvek znalo za njega.

```
sunce.otkrivac=sunce.otkrivac + " i niko"
```

Uradili smo sledeće, uzeli smo podatke o otkrivaču iz instance sunce, na to smo dodali "i niko", i dodelili smo sunce.otkrivac da je otkrivač (u ovom slučaju) "svi i niko". Sada u suncu imamo:

sunce	
tip	zvezda
precnik	1369000
otkrivac	svi i niko

Objektima u okviru novog objekta pristupamo pomoću tačke. Tačka znaci sledeće, u ovoj instanci, nađi instancu koja se zove tako kao što smo napisali.
sunce.tip znači da imamo pristup tipu instance sunce.

Слика 6. Друга лекција - Иницијализација

Оно што је, наравно, неопходно у раду са објектима јесте могућност рада са вредностима података унутар објекта. У Пајтону се користи . (тачка) нотација, где се са тачком после имена инстанце и жељеним елементом долази до вредности елемента те инстанце. Дат је следећи пример:

```
sunce.otkrivac=sunce.otkrivac + " i niko"
```

На овај начин се сада у инстанци `sunce`, па у њеном атрибуту `otkrivac` налази податак `svi i niko`, пошто је прво узет податак који се претходно налазио у оквиру `otkrivac`-а, на њега је надовезано `i niko`, а затим је то све уписано у `otkrivac` ове инстанце `sunce`.

На тренутак је остављено да ученици сами примете да при креирању инстанце постоји један аргумент мање него у функцији за иницијализацију, што до сада није био случај када се радило са функцијама. После показивања промене вредности, на ово је посебно указано. Постоји аргумент мање управо зато што први аргумент при дефинисању функције помаже да се одреди на којој инстанци се обавља рад, а пошто је познато на којој инстанци (у овом случају `sunce`) се ради онда тај аргумент није потребан.

Оно што је уобичајено у већини објектно оријентисаних програмских језика, а није подржано у Пајтону, јесте могућност иницијализације на различите начине. У Пајтону постоји само једна функција за иницијализацију, тј. функција јединственог потписа. Ова ускраћеност се често заобилази помоћу низа аргумената, или помоћу аутоматског постављања вредности [6], на следећи начин:

```
class NebeskaTela(object):

    def __init__(self,tip,precnik,otkrivac="nepoznat"):

        self.tip=tip

        self.precnik=precnik

        self.otkrivac=otkrivac
```

```
sunce=NebeskaTela("zvezda",1369000,"svi")

mesec=NebeskaTela("mesec",123)

print(sunce.otkrivac)

print(mesec.otkrivac)
```

У овом случају излаз ће бити:

```
svi

nepoznat
```

Иако је овај начин заобилажења немогућности више начина иницијализације често коришћен, он има своје недостатке и не може у потпуности заменити тражено. Пошто је курс замишљен за програмере почетнике, овај податак није сматран релевантним.

3.3. Чланске променљиве

Када је потребна променљива која ће постојати без иницијализације и у свим инстанцама имати исту вредност, она се може обезбедити на следећи начин:

```
class NebeskaTela(object):

    galaksija="Mlecni put"

    def __init__(self,tip,precnik,otkrivac):

        self.tip=tip

        self.precnik=precnik

        self.otkrivac=otkrivac
```

Сада ће у свим инстанцама које се направе постојати елемент `galaksija` која ће аутоматски имати постављену вредност `Mlecni put`. Рад са чланским промењивама је као и са осталим, „личним“ променљивама. Приступа јој се помоћу тачка нотације и мења се на исти начин као што је и показано на сајту (слика 7.).

Слика 7. Чланске промењиве

Ученицима је као вежба остављено да направе још инстанци класе небеских тела и да увиде шта се дешава код осталих са галаксијом ако промене њену вредност код само једне инстанце. Оно што ће се десити је да ће вредност код осталих остати као што је подразумевана, што је остављено ученицима да сами открију.

Оно што није приказано у раду са чланским промењивама, због опширности саме лекције и због могуће тешкоће у разумевању приказано је у следећем коду.

```
class NebeskaTela(object):
    galaksija="Mlecni put"
    def __init__(self,tip,precnik,otkrivac):
        self.tip=tip
        self.precnik=precnik
        self.otkrivac=otkrivac
sunce=NebeskaTela("zvezda",1369000,"svi")
mesec=NebeskaTela("mesec",1822,"svi")
sunce.galaksija="ne znam"
print(sunce.galaksija)
print(mesec.galaksija)
NebeskaTela.galaksija="Andromeda"
nepoznato1=NebeskaTela("nn",0,"niko")
nepoznato2=NebeskaTela("nn2",0,"niko");
nepoznato2.galaksija="jos nije pronadjena"
print(sunce.galaksija)
print(mesec.galaksija)
print(nepoznato1.galaksija)
print(nepoznato2.galaksija)
```

Излаз ће бити:

```
ne znam
Mlecni put
```

```
    ne znam

    Andromeda

    Andromeda

    jos nije pronadjena
```

Као што се може приметити, уколико се направе две инстанце, првој промени вредност у елементу `galaksija`, а другој не, добиће се управо оно што је очекивано, промењена вредност само прве инстанце. Ипак, помало неочекиване промене се дешавају при промени вредности `galaksija` целе класе `NebeskaTela`, што се ради на исти начин као и код вредности елемената инстанце (Напомена: на овај начин је могуће мењати вредности само чланских променљивих класе, али не и личних променљивих!) . Код оних инстанци којима није била промењена подразумевана вредност `galaksija`, вредност ће се променити на новопостављену преко саме класе. Код оних код којих је лично промењена вредност, остаће та вредност. Све накнадно направљене инстанце имаће нову подразумевану вредност, све док се и код њих не промени лично. Због ових разлика у раду и могућих потешкоћа у схватању, овај део је изостављен.

4. Методе

Пошто је сврха класа да олакша програмирање и организацију података у објекте, потребне су функције које ће радити тачно оно што се жели са објектима истог типа. Методе представљају функције које се односе само на једну класу и дефинисане су унутар ње. Оне се користе само на инстанцама одређене класе. У оквиру сајта је већ у првој лекцији везаној за класе укратко објашњена поента објектно оријентисаног начина програмирања, а самим тим и шта су методе, тако да се у овој лекцији на сајту не понавља, већ се одмах креће на то шта је потребно од метода у раду са представљеном класом.

Направљене су две методе, `sve` и `citat`, прва која даје све податке о небеском телу и друга која исписује цитат о свемиру. Прва метода је таква да зависи од инстанце, док друга увек исписује исти цитат.

```
class NebeskaTela(object):

    galaksija="Mlecni put"

    def __init__(self,tip,precnik,otkrivac):

        self.tip=tip

        self.precnik=precnik

        self.otkrivac=otkrivac

    def sve(self):

        return "Ovo je %s, precnika %d i otkrio/otkrili su je
                %s!" % (self.tip,self.precnik,self.otkrivac)

    def citat(self):

        return """Postoji teorija koja kaze da ce, ukoliko ikada
                  iko bude otkrio koja je svrha svemira i zasto
                  on postoji, ovaj smesta isceznuti i biti
                  zamenjen necim jos neobicnijim i
                  neobjasnjivijim. Postoji i druga teorija koja
                  tvrdi da se sve to vec desilo."""

sunce=NebeskaTela("zvezda",1369000,"svi")
```

Методе се праве на исти начин као и функције, са једном изменом – увек при дефинисању имају најмање један аргумент, `self`, чије је објашњење већ дато, и који се при позивању функције занемарује.

Даље је дат пример позива ових метода над инстанцом `sunce`, што се ради помоћу тачка нотације. Пошто методе враћају `string` вредност, штампа се вредност која је враћена позивом методе (Слика 8.).

Слика 8. Четврта лекција - Методе

Код претходног кода није постојало дугме копирај, пошто се при извршавању првог дела заправо ништа видно не би догодило, али сада, при штампању, постоји дугме копирај и оно копира претходни код заједно са кодом који одговара дугмету.

У овом делу су рађене једноставне методе из разлога што су функције већ обрађиване раније, па се рачуна на то да ученици разумеју сам концепт функција и да нису потребна додатна објашњења пошто методе функционишу исто као функције, са разликом дефинисања и позивања.

5. Подкласе

Подкласе су концепт који се релативно лако прихвата када је већ прихваћен концепт класа уопште, зато је овде постављено једноставно питање: Шта ако постоје објекти који садрже елементе као већ постојећа класа, али садрже додатне елементе или додатне потребне методе? Логично је да се прављењем нове класе може решити проблем, али зашто правити још једну класу и увећавати код када већ постоји концепт наслеђивања који може помоћи при оваквим случајевима? Подкласе су лепо и логично решење, вероватно идејно настало из живота, које омогућава да се „наследе“ особине већ постојеће класе и њихово унапређивање [7].

Треба да се направи класа планета, која ће бити подкласа класе небеских тела, која ће већ имати одређен тип небеског тела, али додатно, садржи податак око које звезде се окреће и методу која говори другачији цитат.

```
class Planete(NebeskaTela):

    tip="planeta"

    def __init__(self,precnik,otkrivac,zvezda):

        self.precnik=precnik

        self.otkrivac=otkrivac

        self.zvezda=zvezda

    def citat(self):

        return "Jednom kad donesete odluku, citav svemir se
                udruzi da bi se ostvarila."
```

Као што је већ речено на почетку, при дефинисању класе се у заградама налази класа од које се наслеђују својства. Ако се при креирању нове класе жели да она има само основна својства класа уопште, у заградама се налази реч `object`. Међутим, потребно је да нова класа `Planete` наследи својства од класе `Nebeskatela`, па из тог разлога се при дефинисању у заградама налази име класе небеских тела, односно име

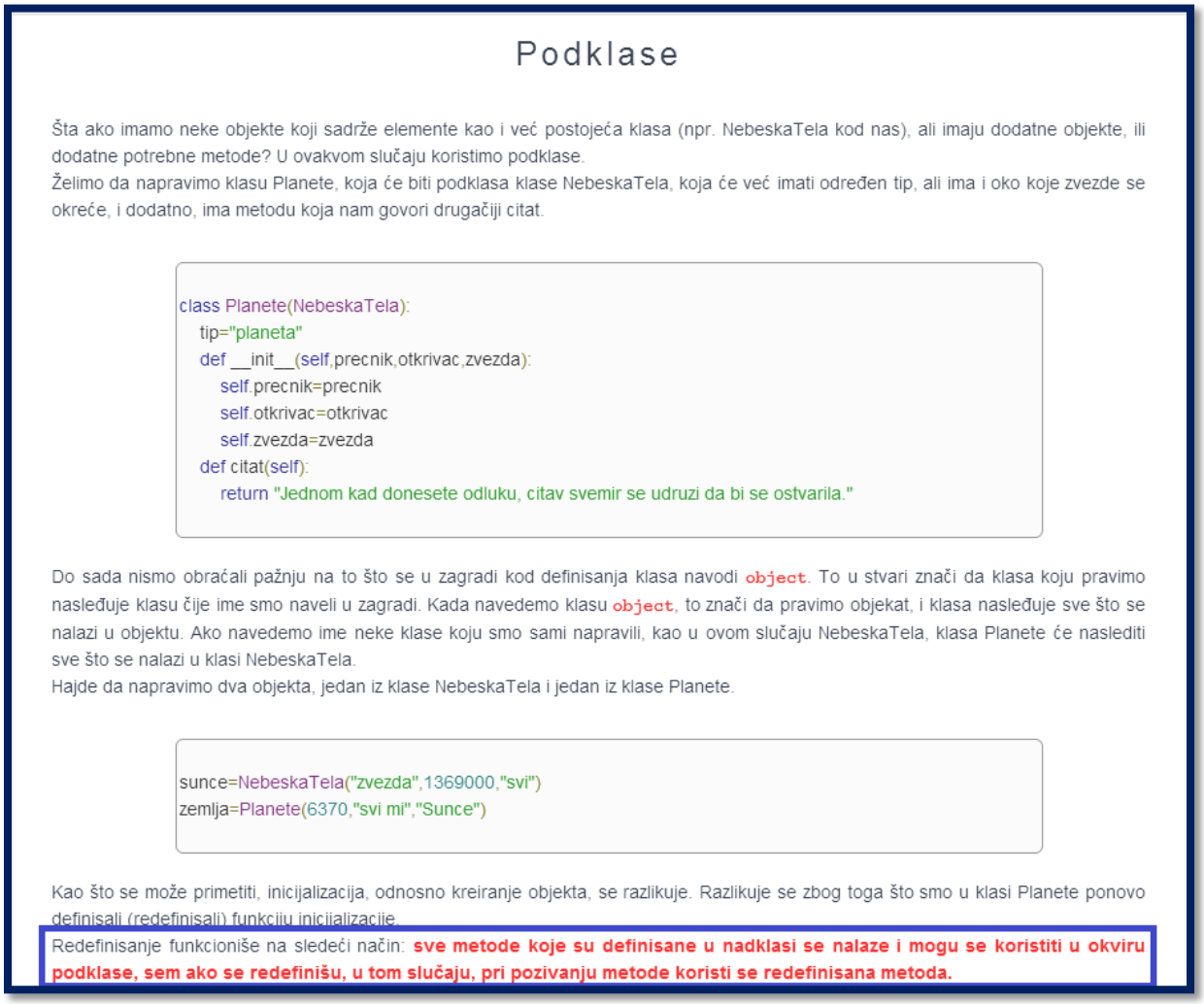
надкласе нове, дефинисане класе. На овај начин се при прављењу објекта класе `Planete` добијају и све методе и објекти надкласе `NebeskaTela`.

Сада се праве два објекта, један класе `NebeskaTela`, а други класе `Planete`.

```
sunce=NebeskaTela("zvezda",1369000,"svi")

zemlja=Planete(6370,"svi mi","Sunce")
```

Оно што се може приметити јесте да се иницијализација разликује. То се десило зато што је функција иницијализације редефинисана тј. поново дефинисана у подкласи `Planete`. Под редефинисањем се подразумева креирање методе са истим именом, али са различитим потписом и/или понашањем. Принцип редефинисања је објашњен и на сајту, назначен црвеним („битно“) словима (Слика 9.).



Podklase

Šta ako imamo neke objekte koji sadrže elemente kao i već postojeća klasa (npr. NebeskaTela kod nas), ali imaju dodatne objekte, ili dodatne potrebne metode? U ovakvom slučaju koristimo podklase.
Želimo da napravimo klasu Planete, koja će biti podklasa klase NebeskaTela, koja će već imati određen tip, ali ima i oko koje zvezde se okreće, i dodatno, ima metodu koja nam govori drugačiji citat.

```
class Planete(NebeskaTela):
    tip="planeta"
    def __init__(self,precnik,otkrivac,zvezda):
        self.precnik=precnik
        self.otkrivac=otkrivac
        self.zvezda=zvezda
    def citat(self):
        return "Jednom kad donesete odluku, citav svemir se udruzi da bi se ostvarila."
```

Do sada nismo obraćali pažnju na to što se u zagradi kod definisanja klasa navodi `object`. To u stvari znači da klasa koju pravimo nasleđuje klasu čije ime smo naveli u zagradi. Kada navedemo klasu `object`, to znači da pravimo objekat, i klasa nasleđuje sve što se nalazi u objektu. Ako navedemo ime neke klase koju smo sami napravili, kao u ovom slučaju NebeskaTela, klasa Planete će naslediti sve što se nalazi u klasi NebeskaTela.
Hajde da napravimo dva objekta, jedan iz klase NebeskaTela i jedan iz klase Planete.

```
sunce=NebeskaTela("zvezda",1369000,"svi")
zemlja=Planete(6370,"svi mi","Sunce")
```

Kao što se može primetiti, inicijalizacija, odnosno kreiranje objekta, se razlikuje. Razlikuje se zbog toga što smo u klasi Planete ponovo definisali (redefinisali) funkciju inicijalizacije.
Redefinisanje funkcioniše na sledeći način: **sve metode koje su definisane u nadklasi se nalaze i mogu se koristiti u okviru podklase, sem ako se redefinišu, u tom slučaju, pri pozivanju metode koristi se redefinisana metoda.**

Слика 9. Подкласе и редефинисање

Разлика између нередефинисане и редефинисане функције се може видети на следећем примеру:

```
print(sunce.sve())

print(zemlja.sve())

print(sunce.citat())

print(zemlja.citat())
```

Након покретања овог кода добиће се:

```
Ovo je zvezda, precnika 1369000 i otkrio/otkrili su je svi!

Ovo je planeta, precnika 6370 i otkrio/otkrili su je svi mi!

Postoji teorija koja kaze da ce, ukoliko ikada iko bude otkrio
        koja je svrha svemira i zasto on postoji, ovaj smesta
        isceznuti i biti zamenjen necim jos neobicnijim i
        neobjasnjivijim. Postoji i druga teorija koja tvrdi da
        se sve to vec desilo.

Jednom kad donesete odluku, citav svemir se udruzi da bi se
        ostvarila.
```

Метода `sve`, која није редефинисана, враћа исти облик стринга (наравно, са другачијим подацима пошто се ради на две различите инстанце), док метода `citat` враћа различите цитате, пошто је редефинисана у новој подкласи. Дакле, све методе које су дефинисане у класи `Nebeskatela`, могу се користити и на инстанцама класе `Planete` и имаће исти утицај, сем ако су редефинисане. Уколико се редефинише метода, на подкласи ће се извршавати редефинисана метода, док ће се на инстанцама оригиналне класе извршавати њена првобитно дефинисана метода. Овај принцип редефинисања већ је на неки начин виђен код креирања методе иницијализације, где је, у ствари, редефинисана општа метода за иницијализацију при креирању нове класе.

Поента дела за вежбу у овој лекцији је увежбавање коришћења подкласа, као и боље схватање логике редефинисања методе и како већ дефинисане методе функционишу са подкласним инстанцама.

5.1. Рад са super функцијом

Често је потребно да се већ постојећи метод из надкласе на неки начин искористи или допуни у подкласи. Уз помоћ `super` функције, може се приступити методи надкласе из подкласе или искористити првобитни метод на инстанци типа подкласе (наравно, ако је првобитно метод редефинисан у подкласи, уколико није редефинисан ова функција нам није потребна) [8]. Супер функција има следећи потпис:

```
super(odKojeKlaseNadklasa,instanca).imeMetode(onoStoSeSaljeMetodi)
```

Примена се може видети на следећем примеру:

```
class Planete(NebeskaTela):

    tip="planeta"

    def __init__(self,precnik,otkrivac,zvezda):

        self.precnik=precnik

        self.otkrivac=otkrivac

        self.zvezda=zvezda

    def sve(self):

        stari=super(Planete,self).sve()

        return stari + " Okrece se oko zvezde " + self.zvezda

zemlja=Planete( 6370,"svi mi","Sunce")

print(zemlja.sve())
```

Напомена: Овај код се мора користити уз део кода за дефинисање класе `NebeskaTela`.

Када се ради са функцијом `super` у оквиру дефинисања нове класе, методи као `instanca`, над којом се врши обрада, даје се аргумент `self`, односно лични аргумент (под којим називом је одређен). У оквиру лекције на сајту, дефинисање функције `super` дато је на мало другачији начин, као што се може видети на слици 10.

Слика 10. `Super` функција и њена примена

То је урађено зато што је тренутно одлучено да се ова функција обрађује само у оквиру подкласног дефинисања, без ванкласног коришћења, па је самим тим непотребно оптерећивање додатним објашњењима.

У методи `sve` сада се догађа следеће, у инстанцу `stari` се уписује стринг који враћа метода `sve` дефинисана у надкласи класе `Planete`, тј. дефинисана у класи `NebeskaTela`. Дакле, метода `sve` из надкласе се примењује на инстанцу. Затим, као резултат враћа се стари резултат на који се додаје информација око које звезде се

окреће дата планета. На овај начин је примењена метода из надкласе у методи из подкласе на инстанци подкласе. Излаз оваквог кода ће бити:

```
    Ovo je planeta, precnika 6370 i otkrio/otkrili su je svi mi!
Okrece se oko zvezde Sunce
```

Овај део курса, нажалост, није подржан у конзоли, па се ученицима истакло да се мора радити у локалној верзији Пајтона, за чије инсталирање и покретање постоји упутство на сајту и доступно је у оквиру сваке лекције.

Коришћење `super` функције ван дефинисања класа је једноставно и може се, уз претходни код из примера, свести на једну линију:

```
    print(super(Planete,zemlja).sve())
```

На овај начин на инстанцу `zemlja` је примењена метода `sve` која се налази у надкласи класе `Planete`. Битно је истаћи да коришћење ове функције није неопходно ако потребна метода није редефинисана у одговарајућој подкласи.

6. Додатно

Део са додацима тренутно обухвата само рад са једном функцијом, али је предвиђено да се прошири и обухвати компликованије и шире концепте у оквиру програмског језика Пајтон.

6.1. Редефинисање штампања објекта

Ученицима је тражено да покрену следећи код:

```
    class NebeskaTela(object):
```

```
        galaksija="Mlecni put"
        def __init__(self,tip,precnik,otkrivac):
            self.tip=tip
            self.precnik=precnik
            self.otkrivac=otkrivac
    sunce=NebeskaTela("zvezda",1369000,"svi")
    print(sunce)
```

Као резултат овог покретања добија се:

```
    <__main__.NebeskaTela object>
```

Овим заправо није много речено о инстанци `sunce` сем да је објекат типа `NebeskaTela`. Уколико је потребна промена тога шта ће функција `print` штампати када јој је прослеђена инстанца одређене класе, то је могуће урадити на следећи начин:

```
    class NebeskaTela(object):
        galaksija="Mlecni put"
        def __init__(self,tip,precnik,otkrivac):
            self.tip=tip
            self.precnik=precnik
            self.otkrivac=otkrivac
        def __repr__(self): #redefinisanje funkcije print (bitno!)
            return "Ovo je %s nebesko telo." % (self.tip)
    sunce=NebeskaTela("zvezda",1369000,"svi")
    print(sunce)
```

Добија се следећи резултат:

```
Ovo je zvezda nebesko telo.
```

Помоћу `__repr__` методе мења се репрезентација инстанци класе небеских тела преко функције `print`. Лекција у оквиру курса на сајту може се видети на слици 11.

Слика 11. Додатно - штампање објекта

Начин штампања инстанце неког објекта може се променити и на други начин који на курсу није приказан, јер није неопходан и јако је сличан већ приказаном начину. То се постиже помоћу методе `__str__`, на исти начин као са методом `__repr__`. Уколико су дефинисане обе методе, `print` функција ће штампати облик из `__str__` методе, а уколико `__str__` није дефинисана, штампаће облик из `__repr__` методе.

Резултат `__repr__` методе се може добити и једноставним позивањем `print repr(t)` кода [9].

6.2. Вишеструко наслеђивање

У Пајтон програмском језику подржано је вишеструко наслеђивање, односно, наслеђивање више класа. Креирање класе која је подкласа две или више класа је једноставно, само при дефинисању класе у загради треба написати све класе које треба да буду наслеђене:

```
class NovaKlasa(staraK1,staraK2):
```

На овај начин, нова класа ће наслеђивати методе и класне инстанце из две старе класе. Јако је битан редослед писања надкласа, јер он одређује приоритет уколико дође до „сударања“ две класе (нпр. због метода истог имена). Прва класа је доминантна класа, од које се наслеђује све, док се од друге класе наслеђује све оно чега нема у првој класи итд. Функција `super` ће позивати методе из прве класе, уколико постоје, а уколико не постоје у првој, прелази на методе у другој класи [10]. Приоритет позивања се може видети на следећем примеру:

```
class Prva(object):
    def rec(self):
        print("prva")
class Druga(object):
   def rec(self):
        print("druga")
    def sledi(self):
        print("Ovo je nezavisno.")
```

```
class Treca(Prva, Druga):

    def stampaj(self):

        super(Treca, self).rec()

        super(Treca, self).sledi()

        print("To je to!")

primerak=Treca()

primerak.stampaj()
```

Излаз ће бити:

```
prva

Ovo je nezavisno.

To je to!
```

Дакле, функција `super` позива методу `rec` из прве класе, а методу `sledi` из друге класе, пошто не постоји у првој.

6.3. Својства (Properties)

Приступање својствима (атрибутима) објекта је у Пајтону јако једноставно и неограничено, својству се може приступити из било ког дела програма помоћу тачка нотације. Ипак, може се десити да је потребно на неки начин ограничити шта се може поставити као вредност објекта, на пример, да пречник небеског тела мора бити број (овај услов нигде није постављен, због својства Пајтон програмског језика), а уколико није, да се постави на нулу. Такође, треба да се то ради при самој иницијализацији објекта, као и при сваком мењају тог својства. То се може постићи као у коду који следи.

```
class NebeskaTela(object):

    galaksija="Mlecni put"

    def __init__(self,tip,precnik,otkrivac):

        self.tip=tip

        self.precnik=precnik

        self.otkrivac=otkrivac

    def set_precnik(self,vred):

        if isinstance(vred,int):

            print("f")

            self._precnik=precnik

        else:

            print("Greska!")

            self._precnik=0

    def get_precnik(self):

        return self._precnik

    precnik=property(get_precnik,set_precnik)
```

Урађено је следеће, помоћу `property` функције написане на крају одређује се које ће две функције преузети враћање својства `precnik` и постављање својства `precnik`. Прва функција ће се позивати кад год се користи вредност `instanca.precnik`, док ће се друга позивати кад год се мења вредност помоћу `instanca.precnik=vrednost` али и при сваком креирању нове инстанце.

Као што се може приметити, име својства унутар `get` и `set` методе разликује се од имена својства заиста (може бити било које, не мора бити `_svojstvo`) зато што би иначе упали у бесконачну петљу, управо зато што се `set` метода позива при сваком

мењању вредности. Помоћу последњег реда је одређено на које се својство (атрибут) односе `get` и `set` методе [11].

На овај начин је ограничено постављање вредности, али тако да се не мења сам начин постављања. Наравно, могле су се направити функције за мењање и постављање вредности, али би у том случају позивање функције на објекат било неопходно, а такође би при креирању објекта све било допуштено, што је била поента избећи.

7. Закључак

Програмски језик Пајтон је врло добар програмски језик за програмере почетнике, што је циљна група управо овог електронског курса. Лак је за учење, пружа много могућности, објектно је оријентисан што је мало сложенији концепт, довољно је близак говорном језику да прелаз не буде тежак, а опет је близак и другим програмским језицима па прелаз са њега на друге програмске језике није проблематичан.

Када се ради о овом курсу, пре свега треба имати на уму да је он намењен програмерима почетницима, које је потребно привући програмирању и начину размишљања програмера. У данашњем времену, када се толико послова обавља на рачунарима, напросто је немогуће не пружити основно програмерско знање што већем броју људи. Управо зато је направљен овај курс - свима доступан, са широко обухваћеним елементима програмирања. Уложен је труд да се направи нешто што ће и људима који се не баве програмирањем (чак и онима који ни немају намеру да се баве) пружити основно знање о програмирању. Уколико се после овог курса одлуче за програмирање, са Пајтона се може лако прећи на друге програмске језике управо због тога што је на пола пута између говорног језика и уобичајених програмских језика. Наравно, овом приликом се не оспорава ни претходни успех најчешће коришћених програмских језика као првих програмских језика који се обично уче у школама.

Објектно оријентисано програмирање у Пајтону, односно рад са класама и објектима, једноставно је и лако имплементирано. Углавном су подржани најчешће коришћени концепти у осталим објектно оријентисаним језицима, а када нису дата је алтернатива да се тај проблем превазиђе. Кроз овај курс су приказани основни

елементи објектно оријентисаног програмирања, рад са класама као што су иницијализација, рад са елементима објекта, креирање и коришћење метода, рад са подкласама, употреба метода и наслеђивања у подкласама као и неке додатне погодности у оквиру Пајтона.

Могуће је да ће се у будућности појавити потреба за креирањем и напредног дела курса о програмском језику Пајтон, зарад оних који желе да се баве програмирањем као и оних којима се свиђа Пајтон као програмски језик уопште. Што се самог Пајтона тиче, постоји још много концепата развијених у њему који се могу приказати и искористити управо за напредне ученике, који тренутно нису приказани на сајту, али ће се можда у будућности појавити у посебном делу. Тренутна жеља је да ученике привуче само програмирање, а уколико би успех био толики да се неко додатно заинтересује за програмирање и Пајтон као програмски језик, курс би сигурно био додатно унапређен.

8. Литература

[1] *Python in the enterprise: Pros and cons – Tech republic*

http://www.techrepublic.com/article/python-in-the-enterprise-pros-and-cons/

Приступљено Јула 2014.

[2] *The Python Programming language*

http://groups.engin.umd.umich.edu/CIS/course.des/cis400/python/python.html

Приступљено Јула 2014.

[3] *Python's History – A Brief History of Python*

http://python.about.com/od/gettingstarted/ss/whatispython_2.htm

Приступљено Јула 2014.

[4] В. Васић, Ј. Марковић, *Пајтон програмски језик*, Универзитет у Новом Саду, октобар 2004.

[5] C. Swaroop*, A Byte of Python*, Enllac web, 2003.

[6] M. Lutz, *Programming Python*, O'Reilly Media, Inc., 2011.

[7] A. Downey, *Think Python*, O'Reilly Media, Inc., 2012.

[8] *Learn Python*

http://learnpythonthehardway.org/book/ex44.html

Приступљено Јула 2014.

[9] Quartz25, Jesdisciple, Hannes Röst, *Python Programming*, Wikibooks, 2012.

[10] W. Chun, *Core python programming*, Prentice Hall Professional, 2001.

[11] *Python Property*

http://www.programiz.com/python-programming/property

Приступљено Августа 2014.