Napredno programiranje i programski jezici

13 Python

Fakultet tehničkih nauka, Novi Sad
23-24/Z
Dunja Vrbaški

```
class Pravougaonik:

   def __init__(self, a, b):
        self.a = a
        self.b = b

   def info(self):
        print(f"Ja sam pravougaonik: {self.a} {self.b}")


   def __str__(self):
        return f"Pravougaonik: a = {self.a}, b = {self.b}"
```

```
Ja sam pravougaonik: 3 5
Pravougaonik: a = 3, b = 5
```

```
p = p.Pravougaonik(3, 5)
p.info()

print(p)
```

```
class Pravougaonik:

    br_instanci = 0

    def __init__(self, a, b):
        self.a = a
        self.b = b
        Pravougaonik.br_instanci += 1
```

```
0
1
```

```
print(pr.Pravougaonik.br_instanci)
p = pr.Pravougaonik(3, 5)
print(pr.Pravougaonik.br_instanci)
```

```
class Pravougaonik:

   br_instanci = 0

    def __init__(self, a, b):
        self.a = a
        self.b = b
        Pravougaonik.br_instanci += 1
```

```
0
1
```

```
x = pr.Pravougaonik

print(x.br_instanci)
p = x(3, 5)
print(x.br_instanci)
```

sve je objekat pa i sama klasa

x je tipa ili klase “tip”
p je tipa ili klase “Pravougaonik”

```python
class Osoba:
    def __init__(self, ime, prezime):
        self.ime = ime
        self.prezime = prezime

    def __str__ (self):
        return f"Osoba: {self.ime} {self.prezime}"

class Student(Osoba):
    def __init__(self, ime, prezime, bri):
        super().__init__(ime, prezime)
        self.bri = bri

    def __str__(self):
        return f"Student: {self.ime} {self.prezime} {self.bri}"



osoba = Osoba("Petar", "Petrovic")
print(osoba)

student = Student("Petar", "Petrovic", 123)
print(student)
```

```
Osoba: Petar Petrovic
Student: Petar Petrovic 123
```

```
class Figura:
    def __init__(self, ime):
        self.ime =  ime

    def povrsina(self):
        pass

class Pravougaonik(Figura):

    def __init__(self, ime, a, b):
        super().__init__(ime)
        self.a = a
        self.b = b

  def povrsina(self):
        return self.a * self.b

    def __str__(self):
        return f"Ja sam pravougaonik ime = {self.ime}, P = {self.povrsina()}"
```

```
Ja sam pravougaonik ime = Pravougaonik 1, P = 2
```

```
import figura as fig

p = fig.Pravougaonik("P1", 1, 2)
print(p)
```

```
from abc import ABC, abstractmethod

class Figura(ABC):
    def __init__(self, ime):
        self.ime =  ime

    @abstractmethod
    def povrsina(self):
        pass

...

class Pravougaonik(Figura):
...
```

```
import figura as fig

p = fig.Pravougaonik("P1", 1, 2)
f = fig.Figura(“F1”)
```

```
TypeError: Can't instantiate abstract class Figura
with abstract method povrsina
```

```
class Pravougaonik(Figura):

    def __init__(self, ime, a, b, id):
        super().__init__(ime)
        self.a = a
        self.b = b
        self._id = id
```

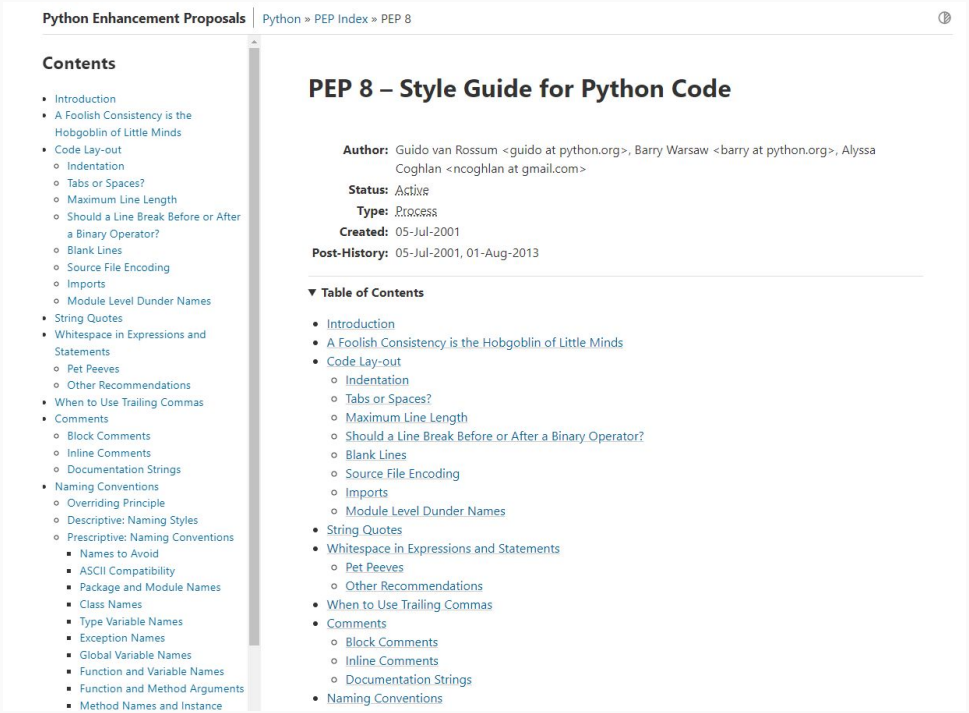
```
p._id += 1
```

In addition, the following special forms using leading or trailing underscores are recognized (these can generally be combined with any case convention):

- `_single_leading_underscore`: weak "internal use" indicator. E.g. `from M import *` does not import objects whose names start with an underscore.
- `single_trailing_underscore_`: used by convention to avoid conflicts with Python keyword, e.g. :

```
tkinter.Toplevel(master, class_='ClassName')
```

- `__double_leading_underscore`: when naming a class attribute, invokes name mangling (inside class FooBar, `__boo` becomes `_FooBar__boo`; see below).
- `__double_leading_and_trailing_underscore__`: "magic" objects or attributes that live in user-controlled namespaces. E.g. `__init__`, `__import__` or `__file__`. Never invent such names; only use them as documented.



Python Enhancement Proposals | Python » PEP Index » PEP 8

Contents

- Introduction
- A Foolish Consistency is the Hobgoblin of Little Minds
- Code Lay-out
  - Indentation
  - Tabs or Spaces?
  - Maximum Line Length
  - Should a Line Break Before or After a Binary Operator?
  - Blank Lines
  - Source File Encoding
  - Imports
  - Module Level Dunder Names
- String Quotes
- Whitespace in Expressions and Statements
  - Pet Peeves
  - Other Recommendations
- When to Use Trailing Commas
- Comments
  - Block Comments
  - Inline Comments
  - Documentation Strings
- Naming Conventions
  - Overriding Principle
  - Descriptive: Naming Styles
  - Prescriptive: Naming Conventions
    - Names to Avoid
    - ASCII Compatibility
    - Package and Module Names
    - Class Names
    - Type Variable Names
    - Exception Names
    - Global Variable Names
    - Function and Variable Names
    - Function and Method Arguments
    - Method Names and Instance

PEP 8 – Style Guide for Python Code

Author: Guido van Rossum <guido at python.org>, Barry Warsaw <barry at python.org>, Alyssa Coghlan <ncoghlan at gmail.com>
Status: Active
Type: Process
Created: 05-Jul-2001
Post-History: 05-Jul-2001, 01-Aug-2013

▼ Table of Contents

- Introduction
- A Foolish Consistency is the Hobgoblin of Little Minds
- Code Lay-out
  - Indentation
  - Tabs or Spaces?
  - Maximum Line Length
  - Should a Line Break Before or After a Binary Operator?
  - Blank Lines
  - Source File Encoding
  - Imports
  - Module Level Dunder Names
- String Quotes
- Whitespace in Expressions and Statements
  - Pet Peeves
  - Other Recommendations
- When to Use Trailing Commas
- Comments
  - Block Comments
  - Inline Comments
  - Documentation Strings
- Naming Conventions

```
5 + 3
print(_)

a = _
print(a)

for _ in range(5):
    print(_)
```

Funkcionalno programiranje

Programska paradigma u kojoj se programi realizuju kroz pozive funkcija. Podaci se prenose kroz ulaz (argumenti) i izlaz funkcija (povratne vrednosti).

Čisto funkcionalno programiranje
Idealano bez modifikacije bilo kakvog stanja i bez bočnih efekata.

!= imperativno programiranje gde je naglasak na promeni stanja

- ne menjamo argumente ili neke globalne promenljive
- iteracija (petlja) → rekurzija
- immutable objekti (promenljive)
- funkcije kao First-Class Citizens

Primer:
- print funkcija
- šta je izlaz, šta menja?

Scheme, Haskel, ML...

Lambda račun

Popularni jezici opšte namene?
Python?

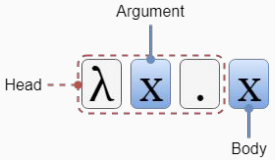
```
def dodaj(lista, v):

    for i in lista:
        lista[i] += v


def dodaj(lista, v):

    nova_lista = []

    for i in lista:
        nova_lista.append(i + v)

    return nova_lista

lista = [0, 1, 2, 3, 4]
nova_lista = dodaj(lista, 100)

print(lista)
print(nova_lista)
```

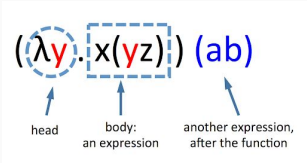
- ne menjamo argumente ili neke globalne promenljive

ranije:

- iteracija (petlja) → rekurzija
- immutable objekti (promenljive)
- funkcije kao First-Class Citizens

- formalna verifikacija, matematički modeli
- modularizacija
- testiranje
- jednostavnije određene programske konstrukcije






```
(define (factorial n)
     (if (< n 2)
          ; Base:
         1
          ; Recurse:
         (* n (factorial (- n 1)))))
```

Iteratori

metod: `__next__()`

```
lista = [1, 2, 3, 4]
it = iter(lista)

it.__next__()
```

```
lista = [1, 2, 3, 4]
it = iter(lista)

it.__next__()
next(it)
```

```
for i in iter(lista):
    print(i)

for i in lista:
    print(i)
```

Generatori

- funkcija + `yield`

```
def generisi_int(n):
      for i in range(n):
         yield i

brojac = generisi_int(5)
a = brojac.__next__()
print(a)

a = next(brojac)
print(a)
```

```
def unazad(n):
    while n > 0:
        yield n
        n -= 1

brojac = unazad(5)
a = brojac.__next__()
print(a)
```

```
def paran(n):
    return ((n % 2) == 0)

lista = [0, 1, 2, 3, 4]
parni = list(filter(paran, lista))
print(parni)
```

```
def kvadrat(n):
    return n**2

lista = [0, 1, 2, 3, 4]
kvadrati = list(map(kvadrat, lista))
print(kvadrati)
```

```
import itertools

iter = itertools.count(3, 5)

i = 0
while i < 10:
    print(next(iter))
    i += 1
```

```
lista = [0, 1, 2, 3, 4]
iter = itertools.cycle(lista)

i = 0
while i < 100:
    print(next(iter))
    i += 1
```