



Γλωσσική Τεχνολογία



Object-Orientation in Python

Everything Is an Object

```
>>> a=[1,2,3]
>>> b=a
>>> b.append(4)
```

- ▶ Τι περιέχουν τα a και b?

```
>>> def hello():
...     print "hello world!"
... 
```

```
>>> def fcall(f):
...     f()
... 
```

```
>>> fcall(hello)
hello world!
```

- ▶ Καλείται η hello...
-



Defining Classes

```
class test:  
    pass
```

- ▶ Ορισμός με το keyword `class`
- ▶ Το σώμα της κλάσης πρέπει να έχει τα κατάλληλα κενά.
- ▶ Ο ορισμός της κλάσης τελειώνει στην πρώτη γραμμή χωρίς τα κενά.

```
class test(parentclass):  
    pass
```

- ▶ Υποστηρίζει κληρονομικότητα
- ▶ Υποστηρίζει πολλαπλή κληρονομικότητα!



Constructor?

- ▶ Δεν υπάρχουν constructors και destructors
- ▶ Κάτι σαν constructor: Η μέθοδος `__init__`
class test:
 def `__init__`(self):
 print "initialized!"
- ▶ Η δήλωση της `__init__` δεν είναι υποχρεωτική.
- ▶ Καλείται κατά την αρχικοποίηση του αντικειμένου, η οποία γίνεται με κλήση της κλάσης με το όνομά της:
`a=test()`
- ▶ Το όρισμα `self` είναι αναφορά στο αντικείμενο (κάτι σαν το `this` της C# και της Java)



__init__ and inheritance

- ▶ Αν δηλωθεί, πρέπει να καλέσει τις `__init__` των γονικών κλάσεων!

```
class ptest1:  
    def __init__(self):  
        print "p1 init!"
```

```
class ptest2:  
    def __init__(self):  
        print "p2 init!"
```

```
class test(ptest1,ptest2):  
    def __init__(self):  
        ptest1.__init__(self)  
        ptest2.__init__(self)
```

- ▶ Αν δεν δηλωθεί στην θυγατρική κλάση, καλείται αυτόματα η `__init__` του πρώτου πατέρα.
 - ▶ Αν δηλωθεί χωρίς να καλέσει τις `__init__` των γονέων, αυτές δεν καλούνται αυτόματα!
-



Memory Management

- ▶ Κανονικό garbage collection
- ▶ Δεν χρειάζεται καταστροφή των αντικειμένων
- ▶ Διατηρείται μετρητής αναφορών στα αντικείμενα
- ▶ Το αντικείμενο καταστρέφεται όταν ο μετρητής γίνει 0
 - >>> a=[1,2,3]
 - >>> b=a
 - >>> b.append(4)
- ▶ Η λίστα που περιέχει τα [1,2,3,4] καταστρέφεται όταν βγουν εκτός εμβέλειας οι αναφορές a και b.
- ▶ Things disappear when no one is looking at them 😊



Methods

▶ Normal

- ▶ Κανονικές συναρτήσεις της κλάσης
- ▶ Καλούνται με χρήση του αντικειμένου, πχ `a.test()`

▶ Private

- ▶ Αν το όνομα μιας μεθόδου αρχίζει με `__` και δεν τελειώνει με `__`, θεωρείται `private` και δεν καλείται από το αντικείμενο.

▶ Special

- ▶ Μέθοδοι που το όνομά τους αρχίζει και τελειώνει με `__`.
- ▶ Δεν καλούνται ρητά, αλλά αυτόματα υπό συγκεκριμένες συνθήκες.



Methods & self

- ▶ Στην δήλωση των μεθόδων το πρώτο όρισμα είναι υποχρεωτικά το `self`
- ▶ Όταν καλείται η μέθοδος, δεν περιλαμβάνεται το `self` στη λίστα των ορισμάτων.
- ▶ Εξαίρεση: Αν κληθεί μέσα από την κλάση μέθοδος της γονικής κλάσης, πρέπει να συμπεριληφθεί το `self`.

```
class ptest:  
    def __init__(self):  
        print "p1 init!"  
    def hi(self,name):  
        print "Parent: Hello "+name+"!"
```

```
class test(ptest):  
    def __init__(self):  
        ptest.__init__(self)  
    def sayhi(self,name):  
        ptest.hi(self,name)
```

```
a=test()  
a.sayhi("vivi")
```



Methods & Inheritance

- ▶ Οι μέθοδοι των γονικών κλάσεων μπορούν να γίνουν `override`.

- ▶ Διαδικασία εύρεσης της μεθόδου που καλείται:

- ▶ Η `python` ψάχνει τη μέθοδο μέσα στην κλάση.
- ▶ Αν δεν τη βρει, ψάχνει μέσα στον πρώτο πατέρα
- ▶ Αν δεν τη βρει, ψάχνει στο δεύτερο πατέρα

- ▶ Κ.Ο.Κ

```
class ptest1:
    def hi(self,name):
        print "Parent1: Hello "+name+"!"
```

```
class ptest2:
    def hi(self,name):
        print "Parent2: Hello "+name+"!"
```

```
class test(ptest1,ptest2):
    pass
```

```
a=test()
a.hi("vivi")
```

- ▶ Θα κληθεί η `hi` της `ptest1` κλάσης
-



Special Methods

- ▶ `__init__`: καλείται στο initialization των objects
- ▶ `__setitem__`: καλείται στην ανάθεση των items μιας συλλογής
- ▶ `__getitem__`: καλείται στην ανάκτηση των items μιας συλλογής

```
class test():  
    def __init__(self):  
        self.data={}  
    def __setitem__(self,key,val):  
        self.data[key]=val  
    def __getitem__(self,key):  
        return self.data[key]
```

```
a=test()  
a["k1"]=1    #καλείται η __setitem__  
print a["k1"]    #καλείται η __getitem__
```



Special Methods - advanced

- ▶ `__repr__`: Καλείται όταν χρησιμοποιούμε τον τελεστή `repr` σε ένα αντικείμενο, ο οποίος επιστρέφει την αναπαράσταση σε `string` του αντικειμένου.
 - ▶ `repr(obj)`
- ▶ `__cmp__`: Καλείται όταν συγκρίνονται δύο αντικείμενα αυτής της κλάσης με τον `==` τελεστή.
 - ▶ `obj1==obj2`
- ▶ `__len__`: Καλείται όταν χρησιμοποιείται σε αντικείμενο ο τελεστής `len`
 - ▶ `len(obj)`
- ▶ `__delitem__`: Καλείται όταν χρησιμοποιούμε τον `del` τελεστή για να διαγράψουμε ένα `item` από μια συλλογή
 - ▶ `del obj[key]`



Data Attributes

- ▶ Data attributes (=Instance variables)
 - ▶ Μεταβλητές (μνήμη) που ανήκει στο instance
- ▶ Αρχικοποιούνται μόλις πάρουν τιμή σε κάποια συνάρτηση
- ▶ Καλό είναι όλες να αρχικοποιούνται στην `__init__`
- ▶ Για να δηλωθεί ότι είναι data attribute και όχι local μεταβλητή της συνάρτησης, υποχρεωτικά χρησιμοποιείται το `self`

```
class test():  
    def __init__(self):  
        self.x=0  
    def incr(self,num):  
        self.x+=num
```

```
a=test()  
a.incr(3)  
print a.x      #εκτυπώνει 3
```



Class Attributes

- ▶ **Class attributes (=Static variables)**

- ▶ Μεταβλητές (μνήμη) που ανήκει στην κλάση.

```
class test():  
    cnt=0 #class attribute  
    def __init__(self):  
        self.inst_cnt=0 #data attribute
```

```
test.cnt+=2  
print test.cnt #ΕΚΤΥΠΩΝΕΙ 2  
a=test()  
a.inst_cnt+=1  
print a.inst_cnt #ΕΚΤΥΠΩΝΕΙ 1  
print a.cnt #ΕΚΤΥΠΩΝΕΙ 2
```



Everything Is an Object: So what?

- ▶ Η Python προσφέρει ένα σύνολο συναρτήσεων που διαχειρίζονται τις κλάσεις και τις συναρτήσεις σαν να ήταν αντικείμενα.
 - ▶ The Power of Introspection (Dive into Python κεφ. 4)
- ▶ Μπορείτε να κάνετε πράγματα που σε άλλες γλώσσες είναι δύσκολο ή αδύνατο να γίνουν!

▶ Πχ αυτό:

```
def incr(num,step):  
    return num+step  
def decr(num,step):  
    return num-step
```

```
print "Give initial number:"  
num=int(raw_input())  
print "Give step:"  
step=int(raw_input())  
if(num<step):  
    f=incr  
else:  
    f=decr  
res=f(num,step)  
print res
```



Functions for Introspection

▶ type

- ▶ Επιστρέφει τον τύπο οποιουδήποτε αντικειμένου

```
>>> type(1)
<type 'int'>
>>> type([])
<type 'list'>
>>> def foo():
...     pass
...
>>> type(foo)
<type 'function'>
```

▶ dir

- ▶ Επιστρέφει λίστα με τα ονόματα των μεθόδων οποιουδήποτε αντικειμένου

```
>>> dir([])
['_add__', '_class__', '_contains__', '_delattr__', '_delitem__', '_delslice__', '_doc__', '_eq__', '_format__',
'_ge__', '_getattr__', '_getitem__', '_getslice__', '_gt__', '_hash__', '_iadd__', '_imul__',
'_init__', '_iter__', '_le__', '_len__', '_lt__', '_mul__', '_ne__', '_new__', '_reduce__', '_reduce_ex__',
'_repr__', '_reversed__', '_rmul__', '_setattr__', '_setitem__', '_setslice__', '_sizeof__', '_str__',
'_subclasshook__', 'append', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

▶ getattr

- ▶ Επιστρέφει αναφορά σε συνάρτηση
- ▶ Μπορεί να χρησιμοποιηθεί το όνομά της σε string για να κληθεί

```
>>> a=[1,2]
>>> getattr(a,'append')(3)
>>> a
[1, 2, 3]
```

