Python Tutorial Documentation

Release 0.0.1

Panagiotis Mavrogiorgos

February 17, 2017

Contents

Author Panagiotis Mavrogiorgos

Source code http://bitbucket.org/pmav99/python_tutorial

Generated February 17, 2017

Version 0.0.1

CHAPTER 1

Contents:

Αντικειμενοστραφ Προγραμματισμ (Object-Oriented Programming)

Προγραμματιστικ παραδεγματα

Note: Η εντητα αυτ εναι περισστερο πληροφοριακ. Το "ζουμ" ξεκινει απ την επμενη.

Ο Αντικειμενοστραφ Προγραμματισμ (Object-Oriented Programming OOP για συντομα), πω λει και το νομ του συνδεται μεσα με τα αντικεμενα και κατ συνπεια, πω θα δομε και στη συνχεια, και με τι κλσει. Τι εναι μω ο Αντικειμενοστραφ Προγραμματισμ;

Κατ' ουσαν, ο OOP εναι να τρπο οργνωση των προγραμμτων που γρφουμε. Ο τρπο αυτ οργνωση, δεν εναι φυσικ οτε μοναδικ, οτε καν ο βλτιστο. λλοι τρποι οργνωση εναι ο διαδικαστικ (procedural imperative) και ο συναρτησιακ (functional). Συνηθζεται, αυτο οι τρποι οργνωση τεχνικ οργνωση των προγραμμτων να ονομζονται προγραμματιστικ παραδεγματα (programming paradigms).

Η επιλογ του προγραμματιστικο παραδεγματο το οποο θα χρησιμοποισουμε εξαρτται απ το πρβλημα το οποο καλομαστε να επιλσουμε, αλλ και απ τη γλσσα προγραμματισμο την οποα χρησιμοποιομε. Δεν επιτρπουν λε οι γλσσε προγραμματισμο τη χρση λων των προγραμματιστικν παραδειγμτων. Π.χ. υπρχουν γλσσε που επιτρπουν τη δημιουργα μνο διαδικαστικν προγραμμτων πω η Fortran. Υπρχουν λλε που επιτρπουν τη δημιουργα μνο (κυρω) συναρτησιακν προγραμμτων, πω η Lisp και οι πολλ παραλλαγ τη. Εν υπρχουν και γλσσε που επιτρπουν, με περισστερη η λιγτερη ευκολα, να εφαρμσει περισστερα απ να προγραμματιστικ παραδεγματα. Η Python ανκει σε αυτν την κατηγορα καθ σου επιτρπει να γρψει κδικα που χρησιμοποιε και τα τρα προγραμματιστικ παραδεγματα.

Note: Το πιο χαρακτηριστικ σω παρδειγμα προβλματο που προσφρεται για λση μσω αντικειμενοστραφο προγραμματισμο εναι ο σχεδιασμ GUI.

Τα πντα εναι αντικεμενα (Everything is an object)

Μα απ τι πλον συνθει εκφρσει στα κεμενα που αναφρονται στην Python εναι τι "τα πντα εναι αντικεμενα" (everything is an object). Ευθ αμσω φυσικ, μσα απ την φρση αυτ ξεπηδον δο ερωτματα:

- Τι εναι τα αντικεμενα (objects);
- Πω δημιουργονται τα αντικεμενα;

Το πρτο ερτημα θα το απαντσουμε στην επμενη εντητα. Το δετερο ερτημα μπορε μω να απαντηθε πολ πολ συντομα. Τα αντικεμενα (objects) δημιουργονται απ τι κλσει (classes). Για την ακρβεια μλιστα, οι κλσει ορζονται ω εργοστσια αντικειμνων (object factories). Ενα δηλαδ, τα στοιχεα εκενα τη γλσσα, τα οποα κατασκευζουν αντικεμενα (objects). Προκειμνου ββαια να γνει καλτερα κατανοητ αυτ θα πρπει πρτα να δομε τι εναι τα αντικεμενα.

Αντικεμενα (object)

Προκειμνου να εξηγσουμε τον ρο αντικεμενο (object) στην Python συχν βοηθει να σκεφτμαστε τα αντικεμενα (objects) ω κτι ανλογο με αυτ που ονομζουμε στη γραμματικ τη φυσικ γλσσα "ουσιαστικ".

Για να γνει πιο σαφ αυτ α σκεφτομε ορισμνα ουσιαστικ. Καρκλα, τραπζι, σκλο, γτα, κκλο, ορθογνιο, οδηγ εναι ορισμνα που πιθαν ρχονται στο μυαλ. Αν προσπαθσουμε να δομε το κοιν σημεο λων των παραπνω θα δομε τι πρκειται για:

> Οντττε (μψυχε ψυχε) οι οποε χουν συγκεκριμνε ιδιτητε /και μπορον να εκτελον συγκεκριμνε ενργειε.

Α δομε μερικ παραδεγματα:

- Μα καρκλα χει τι ιδιτητε χρμα, ψο, υλικ κτλ.
- να ορθογνιο αντστοιχα χει τι ιδιτητε πλτο, ψο, εμβαδν, περμετρο κτλ.
- Μα γτα χει τι ιδιτητε νομα, ηλικα, φλλο κτλ αλλ επση μπορε και να εκτελε διφορε ενργειε πω π.χ. να νιαουρσει, να τρξει, να περπατσει, να σκαρφαλσει, να φει κτλ.
- να οδηγ μπορε να προβε σε διφορε ενργειε, πω πχ να στρψει, να φρενρει, να επιταχνει, να βλει μπροστ τη μηχαν κτλ.

πω γνεται σαφ απ τα παραπνω, σε ρου φυσικ γλσσα, οι ιδιτητε των αντικειμνων εναι λλα ουσιαστικ, εν οι ενργειε που μπορον να εκτελσουν εναι ρματα.

Note: Την αντιστοχιση αυτ μεταξ ιδιοττων - ουσιαστικν και ενεργειν - ρημτων καλ εναι να την κρατσουμε στο νου μα καθ θα μα χρειαστε αργτερα ταν θα δομε πω μπορομε να χρησιμοποισουμε τι κλσει.

Ο παραπνω "ορισμ" των αντικειμνων εναι φυσικ πολ γενικ και δσκολα θα ντεχε σε ενδελεχ εξταση απ ναν φιλλογο. Παρλα αυτ, εναι να ορισμ που μα βολεει ιδιατερα ταν επιστρφουμε στον κσμο του Αντικειμενοστραφο Προγραμματισμο και αυτ γιατ τα αντικεμενα (objects) εναι εκενε οι "οντττε" οι οποε μα επιτρπουν να περιγρφουμε κθε τι που χει ιδιτητε και μπορε να εκτελε ενργειε. Με λγα λγια δηλαδ, μσω των αντικειμνων μπορομε να περιγρψουμε πρακτικ σχεδν οτιδποτε συναντμε στον υλικ κσμο.

Χρησιμοποιντα την ορολογα τη Python, οι ιδιτητε εν αντικειμνου ονομζονται **attributes**, εν οι ενργει που μπορε να εκτελσει ονομζονται **methods** (μθοδοι).

Κλσει (Classes)

> **Warning:** Η σνταξη των κλσεων στην Python εναι αρκετ απλ. Παρλα αυτ, σε πρτη φση θα χρησιμοποισουμε μια ακμη πιο απλοποιημνη σνταξη (η οποα μω μοιζει αρκετ με Python) προκειμνου να καταλβουμε ευκολτερα ορισμνε βασικ ννοιε του Αντικειμενοστραφο Προγραμματισμο.
> Την κανονικ σνταξη τη Python θα την δομε στη συνχεια.

Μια κλση δεν εναι τποτα λλο παρ να **o**ρισμ. Αυτ που ορζει εναι το ποιε ιδιτητε (attributes) και ποιε μεθδου (methods) θα χει να αντικεμενο.

Α δομε να παρδειγμα. Σε πρτη φση α φτιξουμε μια κλση που ορζει να Αντικεμενο Γτα, (με λλα λγια δηλαδ μια Γτα!):

```
class Cat():
    name
    age
    sex

    def eat():
        print("%s is eating" % name)

    def sleep():
        print("%s is sleeping" % name)
```

Η πρτη γραμμ εναι αυτ στην οποα ορζεται το νομα τη κλση. Προσξτε την παρουσα των παρενθσεων. Τη χρση του θα τη δομε στη συνχεια. Οι γραμμ που ακολουθον, αποτελον το σμα τη κλση (class body). Στη συγκεκριμνη περπτωση, ορσαμε τι οι γτε που θα δημιουργηθον απ την κλση αυτ, θα χουν 3 ιδιτητε (νομα, ηλικα και φλο) και θα μπορον να εκτελον 2 ενργειε (θα χουν 2 μεθδου δηλαδ) , να τρνε και να κοιμονται.

λε οι γτε που θα δημιουργηθον απ την κλση αυτ θα χουν μνο αυτ τι ιδιτητε (attributes) και θα μπορον να εκτελον μνο τι συγκεκριμνε ενργειε.

Note: Θα μπορσαμε φυσικ να κνουμε την κλση μα πολ πιο σνθετη. Μια γτα εξλλου εναι να πολ σνθετο οργανισμ... Αλλ εδ για διδακτικο λγου προτιμσαμε να κρατσουμε τα πργματα απλ. Στην πρξη, συνθω, οι κλσει μα θα εναι αρκετ πιο σνθετε.

Τρα θα χρησιμοποισουμε την κλση που ορσαμε παραπνω για να δημιουργσουμε δο νε γτε (δηλαδ δο να αντικεμενα Γτα). Τη γτα του Joe, μια θηλυκ γτα δο ετν που τη λνε "Kitty" και τη γτα τη Mary, μια αρσενικ γτα οκτ ετν που τη λνε "Paul":

```
joes_cat = Cat(name="Kitty", age=2, sex="female")
marys_cat = Cat(name="Paul", age=8, sex="male")
```

Η σνταξη για τη δημιουργα των νων αντικειμνων Γτα εναι πολ απλ. Απλ χρησιμοποιομε το νομα τη κλση και μσα στι παρενθσει δνουμε τι τιμ των ιδιοττων (attributes) τη κλση. Με τον τρπο αυτ, δνοντα δηλαδ διαφορετικ τιμ στα attributes τη κλση, μπορομε να δημιουργσουμε πολλ, διαφορετικ μεταξ του γτε. Δεν υπρχει καννα περιορισμ στον αριθμ των νων αντικειμνων που θα δημιουργσουμε απ μα κλση. Μπορομε να δημιουργσουμε σα να αντικεμενα θλουμε.

λα τα να αντικεμενα θα χουν ακριβ τι διε ιδιτητε (attributes) και τι διε μεθδου. Οι τιμ των ιδιοττων του μω θα εναι διαφορετικ μεταξ του. Μπορομε φυσικ να δσουμε ακριβ τι διε τιμ στι ιδιτητε, οπτε θα δημιουργηθον δο μοια αντικεμενα, αλλ συνθω δεν χουμε λγο να κνουμε κτι ττοιο.

Note: Υπενθυμζουμε τι προηγουμνω ορσαμε τι κλσει σαν εργοστσιο αντικειμνων (factory object). Ακριβ πω να εργοστσιο που φτιχνει καρκλε μπορε να κατασκευσει καρκλε διαφορετικο σχεδου, διαστσεων και χρματο σε ποια ποστητα επιθυμε, τσι και οι κλσει μπορον να κατασκευσουν αντικεμενα με διαφορετικ ιδιτητε σε ποια ποστητα επιθυμομε.

Στην ορολογα του Αντικειμενοστραφο Προγραμματισμο, λα τα να αντικεμενα που δημιουργονται απ μα κλση ονομζονται στιγμιτυπα (instances).

Note: πω μα φωτογραφα εν αθλητ που τρχει αποτελε την απεικνιση μα μνο απ λε τι θσει που πρασε κατ τη διρκεια του αγνα του, τσι και μα instance αποτελε μα μνο αποτπωση λων των δυνατν συνδυασμν που μπορον να χουν οι ιδιτητε μια κλση.

Πρσβαση σε ιδιτητε και μεθδου

Προκειμνου να αποκτσουμε πρσβαση στι ιδιτητε και στι μεθδου των αντικειμνων που δημιουργσαμε, χρησιμοποιομε το λεγμενο *dot notation*. Πχ. για να εκτυπσουμε στην οθνη το νομα τη κθε γτα θα δναμε:

```
print(joes_cat.name)
print(marys_cat.name)
```

Το αποτλεσμα τη εκτλεση του παραπνω κδικα θα εναι:

```
Kitty
Paul
```

Αντστοιχα για να πομε στη γτα του Joe να κοιμηθε και στη γτα τη Mary να φει θα το κναμε ω εξ:

```
joes_cat.sleep()
marys_cat.eat()
```

Το αποτλεσμα τη εκτλεση του παραπνω κδικα θα εναι:

```
Kitty is eating.
Paul is sleeping.
```

Περισστερα για τι Μεθδου

Αν προσξουμε τον ορισμ των μεθδων μσα στο σμα τη κλση, θα δομε τι δε διαφρουν απ τον ορισμ των συναρτσεων. Μα μθοδο δεν εναι τποτα λλο παρ μα συνρτηση που ανκει σε να αντικεμενο. λα σα ξρουμε για τι τυπικ συναρτσει τη Python ισχουν και εδ. Μπορομε να δσουμε ξτρα ορσματα (arguments) σε μα μθοδο κτλ. Πχ α ξαναορσουμε την κλση τη Γτα, βζοντα αυτ τη φορ υποχρεωτικ ορσματα στι μεθδου τη:

```
class Cat():
    name
    age
    sex

    def eat(food):
        print("%s is eating %s." % (name, food))

    def sleep(time):
        print("%s is sleeping for %d minutes." % (name, time))
```

Αυτ τη φορ λοιπν, θα μπορομε να πομε στι γτε που θα δημιουργσουμε τι να φνε και πση ρα να κοιμηθονε. Πχ με τον ακλουθο κδικα, θα δημιουργσουμε μια γτα στην οποα θα πομε πρτα να φει ποντκια, στη συνχεια να κοιμηθε για 120 λεπτ και μετ να πιει γλα:

```
alley_cat = Cat(name="Leo", age=4, sex="male")

alley_cat.eat("mice")
alley_cat.sleep(120)
alley_cat.eat("milk")
```

Το αποτλεσμα τη εκτλεση του παραπνω κδικα θα εναι:

```
Leo is eating mice.
Leo is sleeping for 120 minutes.
Leo is eating milk.
```

Note: Παρ τι επμονε προσπθει του, ο συγγραφα ποτ δεν κατφερε να πει στη γτα του πση ρα να κοιμηθε...

Βλπετε, στι κλσει που ορζουμε εμε, μπορομε να αποφασσουμε για τη συμπεριφορ των αντικειμνων που θα δημιουργηθον. Στον πραγματικ κσμο πλι χι...

Warning: Το τι οι μθοδοι εναι ακριβ δι με τι συναρτσει δεν εναι απλυτα ακριβ. Στην ψευδογλσσα που χρησιμοποιομε, ισχει μεν κτι ττοιο αλλ στην Python οι μθοδοι χουν μα διαφορ απ τι συναρτσει. Για την ρα δεν εναι σημαντικ. Απλ κρατστε το στο μυαλ σα.

Κληρονομικτητα (Inheritance)

σω η πλον κεφαλαιδη νοια του Αντικειμενικοστραφο Προγραμματισμο εναι η κληρονομικτητα (inheritance).

Με τον ρο κληρονομικτητα, εννοομε τη δυναττητα που χει μια κλση να κληρονομε λε τι ιδιτητε και τι μεθδου μια λλη κλση. Χρησιμοποιντα λγο πιο επσημη ορολογα, λμε τι η κλση που ορζεται πρτη εναι η βασικ κλση (base class) και η κλση που κληρονομε τη βασικ κλση ονομζεται παργωγη κλση (derived class). Εναλλακτικ οι βασικ κλσει ονομζονται και υπερκλσει εν οι παργωγε κλσει ονομζονται υποκλσει.

Α δομε να παρδειγμα:

```
class BaseClass():
    attr1
    attr2

    def method1():
        print("You just called method1.")

    def method2():
        print("You just called method2.")

class DerivedClass(BaseClass):
    pass
```

Η υπερκλση (`BaseClass`) δεν χει καμα διαφορ απ τι κλσει που χουμε δει ω τρα. Η υποκλση (`DerivedClass`) μω χρησιμοποιε ελαφρ διαφορετικ συνταξη. Αντ οι παρενθσει τη πρτη γραμμ του ορισμο τη να εναι κεν:

```
class DerivedClass():
    pass
```

περιχουν το νομα τη υπερκλση:

```
class DerivedClass(BaseClass):
    pass
```

Αυτ η μικρ διαφορ στη συνταξη κνει λη τη διαφορ! Με τον τρπο αυτ, οι instances τη `DerivedClass` αποκτον λε τι ιδιτητε και λε τι μεθδου που ορστηκαν στην `BaseClass`! Α δομε να παρδειγμα. Θα δημιουργσουμε δο instances τη `DerivedClass` και θα καλσουμε τι μεθδου που χουν οριστε στην `BaseClass`:

```
# Class Instantiation
instance1 = DerivedClass(attr1="value1", attr2="value2")
instance2 = DerivedClass(attr1="value3", attr2="value4")

instance1.method1()
instance2.method2()
```

Το αποτλεσμα του παραπνω κδικα θα εναι:

```
You just called method1.
You just called method2.
```

πω βλπουμε, αν και το class body τη `DerivedClass` εναι κεν, παρλα αυτ, τα στιγμιτυπ τη, τα αντικεμενα δηλαδ που δημιουργονται απ την κλση, μπορον να καλον κανονικ τι μεθδου τη `BaseClass`.

Note: Δηλαδ, προσθτοντα το νομα τη υπερκλση στον ορισμ τη κλση, (μσα στι παρενθσει τη πρτη γραμμ) ο ορισμ τη `DerivedClass` γνεται ισοδναμο με τον ακλουθο:

```
class DerivedClass():
    attr1
    attr2

    def method1():
        print("You just called method1.")

    def method2():
        print("You just called method2.")
```

Επειδ το παραπνω εναι πολ γενικ, α δομε και να πιο χειροπιαστ παρδειγμα. Θα ορσουμε λοιπν μα κλση η οποα θα δημιουργε Ζα:

```
class Animal():
    species
    race
    sex

    def speak():
        print("I don't know what to say...")
```

Στη συνχεια θα ορσουμε δο κλσει που εξειδικεουν την κλση Ζου:

```
class Cat(Animal):
    def speak():
        print("Meow!")

class Dog(Animal):
    def speak():
        print("Woof!")
```

Οι κλσει Σκλου και Γατα κληρονομον τι ιδιτητε και τι μεθδου τη κλση Ζου. Προσοχ στη μθοδο `speak()` μω! πω βλπουμε και οι δο υποκλσει ξαναορζουν την μθοδο που κληρονμησαν απ την υπερκλση του. Αυτ εναι μα απ τι βασικτερε τεχνικ τη Κληρονομικτητα (Inheritance). Οι υποκλσει δεν κληρονομον απλ ιδιτητε (attributes) και μεθδου (methods), αλλ μπορον να ξαναορσουν τι μεθδου που κληρονμησαν, εξειδικεονται.

Α το δομε και στην πρξη:

```
# Class Instances
my_dog = Dog()
my_cat = Cat()

my_dog.speak()
my_cat.speak()
```

Το αποτλεσμα του παραπνω κδικα εναι:

```
Woof!
Meow!
```

Φυσικ οι υποκλσει, εκτ απ το να ξαναορσουν τι μεθδου που κληρονομον, μπορον να ορσουν και νε μεθδου. Παρδειγμα δε θα δομε για αυτ γιατ εναι μλλον απλ.

Απλ vs Πολλαπλ Κληρονομικτητα (Single vs Multiple Inheritance)

Αργ γργορα (μλλον γργορα) θα ακοσετε τον ρο Πολλαπλ Κληρονομικτητα (Multiple Inheritance). Η κληρονομικτητα που χουμε δει ω τρα εναι η λεγμενη Απλ Κληρονομικτητα (Single Inheritance). Η διαφορ μεταξ τη απλ και τη πολλαπλ κληρονομικτητα γκειται στον αριθμ των υπερκλσεων που χει μα συγκεκριμνη υποκλση. Αν κληρονομε απ μα μνο υπερκλση ττε μιλμε για απλ κληρονομικτητα. Αν κληρονομε απ δο περισστερε υπερκλσει, ττε μιλμε για πολλαπλ.

να παρδειγμα πολλαπλ κληρονομικτητα εναι το ακλουθο:

```
class BaseClass1():
    pass

class BaseClass2():
    pass

class DerivedClass(BaseClass1, BaseClass2):
    pass
```

Warning: Η πολλαπλ κληρονομικτητα εναι περπλοκη. Το παρδειγμ μα δεν αναδεικνει τη δυσκολα τη, αλλ πιστψτε με, δεν εναι εκολο να την κνει να συμπεριφερθε σωστ. Η χρση τη πολλαπλ κληρονομικτητα εναι συνθω νδειξη κακο design. Αυτ εναι και ο λγο που υπρχουν πολλ γλσσε προγραμματισμο που υποστηρζουν μνο απλ κληρονομικτητα (πχ Java). Στην πλειοψηφα των περιπτσεων, υπρχει απλοστερο τρπο να κνουμε αυτ που θλουμε απ το να καταφγουμε στην πολλαπλ κληρονομικτητα. Καλ εναι να την αποφεγετε (εκτ και αν ξρετε τι κνετε, οπτε μλλον δε θα διαβζετε αυτ το κεμενο :P).

Στο σημε αυτ, οφελουμε να διασαφηνσουμε τι μα αλληλουχα κλσεων που διαδοχικ κληρονομον η μα την λλη δεν εναι πολλαπλ κληρονομικτητα. Παραδεγματο χρη το ακλουθο παρδειγμα εναι απλυτα τυπικ απλ κληρονομικτητα:

```
class Animal():
    pass

class Mamal(Animal):
    pass

class Feline(Mamal):
    pass

class Cat(Feline):
    pass
```

Θα μπορουσαμε φυσικ να χουμε πολ περισστερε απ τσσερι κλσει. Τα γενεαλογικ δντρα του ζωικο και του φυτικο βασιλεου αποτελον πολ χαρακτηριστικ παραδεγματα ττοιου εδου αλληλουχιν.

Πτε χρησιμοποιομε την Κληρονομικτητα;

Το θεμελιδε αυτ ερτημα, ευτυχ, χει πρα πολ απλ απντηση. Η (απλ) *Κληρονομικτητα* χρησιμοποιεται ταν χουμε μια ακολουθα (η ιεραρχα αν προτιμτε) αντικειμνων τα οποα πηγανουν απ το γενικτερο στο ειδικτερο. Το παρδειγμα με τη προηγομενη εντητα (Ζα Θηλαστικ Αιλουροειδ Γτε) εναι πολ χαρακτηριστικ.

Αν μελετσουμε τι σχσει μεταξ των διαδοχικν κλσεων θα δομε τι η "κθε υποκλση εναι η υπερκλση τη". Δηλαδ:

> Μα Γτα εναι Αιλουροειδ.
>
> να Αιλουροειδ εναι Θηλαστικ.
>
> να Θηλαστικ εναι Ζο.

Η σχση αυτ ισχει χι μνο για την αμσω αντερη υπερκλση, αλλ για κθε υπερκλση. Δηλαδ:

> Μα Γτα εναι Αιλουροειδ.
>
> Μια Γτα εναι και Θηλαστικ.
>
> Μια Γτα εναι και Ζο.

Κθε φορ που χουμε μια παρμοια σχση μεταξ αντικειμνων, δηλαδ μπορομε να πομε τι κτι εναι κτι λλο (is-a relationship) ττε μπορομε/πρπει να χρησιμοποισουμε κληρονομικτητα. Αυτ μω δεν απαντει στο γιατ να το κνουμε αυτ... Γιατ να μην ορσουμε κατευθεαν την κλση τη Γτα; Γιατ να μπερδευμαστε με κλσει, υποκλσει κτλ κτλ;

Η απντηση εναι η εξ. Αν χουμε να ασχοληθομε μνο με αντικεμενα Γτα ττε δεν μα χρειζονται λα αυτ. Αν μω χουμε να κατασκευσουμε Γτε, Λιοντρια, Τγρει κτλ ττε τα οφλη αρχζουν να φανονται. Ορζουμε αρχικ τα κοιν στοιχεα λων αυτν των κλσεων στην υπερκλση Αιλουροειδ, την κληρονομομε και την εξειδικεουμε καταλλλω στι υποκλσει.

Αν μλιστα εκτ απ Αιλουροειδ χουμε να κατασκευσουμε και αντικεμενα Σκλου, Αλεπο, Λκου, αλλ και Αρκοδα, Αλγου, Φκια κτλ ττε η κλση Θηλαστικ κατευθεαν αποκτει νημα.

Με τον διο τρπο, αν στο πργραμμ μα χρειαζμαστε και Ψρια ντομα κτλ ττε η υπερκλση Ζο βρσκει και αυτ τη θση τη. Η κθε μα απ τι κλσει Ψρι, ντομο κτλ θα χει φυσικ το δικ τη ιεραρχικ δντρο, στο οποο θα εξειδικεεται καταλλλω.

Note: Συνοψζοντα, ταν για δο αντικεμενα *A* και *B* μπορομε να πομε τι τ**ο A** εναι τ**ο B**, ττε μπορομε να χρησιμοποισουμε κληρονομικτητα.

Σνθεση (Composition)

Τι εναι η Σνθεση;

Η Σνθεση (Composition) εναι η δετερη τεχνικ του OOP που θα αναπτξουμε.

πω θα θυμστε, εχαμε αναφρει προηγουμνω τι υπρχει μια αντιστοιχα μεταξ των "ιδιοττων" (attributes) μια κλση και των "ουσιαστικν" τη φυσικ γλσσα. Εκτ απ αυτ μω, αναφραμε τι και τα δια τα αντικεμενα (objects) εναι "ουσιαστικ".

Συνδυζοντα τι δο παραπνω προτσει προκπτει τι:

> **οι ιδιτητε (attributes)** εν αντικειμν**ου** εναι και αυτ με τη σειρ τ**ου** αντικεμενα**!**

Η παρατρηση αυτ εναι η βασικ ιδα πσω απ την τεχνικ τη Συνθεση.

Η Συνθεση στην πρξη

Α δομε να παρδειγμα. Α σκεφτομε μια μηχαν αυτοκιντου. Η μηχαν αυτ αποτελεται απ πολλ εξαρτματα. Κλινδροι, βαλβδε, πιστνια, μπουζ εναι μερικ μνο απ αυτ. Η σχση που συνδει την μηχαν με τα εξαρτματ τη εναι η εξ:

> Η *Μηχαν* χει του κυλνδρου.
>
> Η *Μηχαν* χει τι βαλβδε.
>
> Η *Μηχαν* χει τα πιστνια.

Δηλαδ, η μηχαν χει εξαρτματα-ιδιτητε, το καθνα απ τα οποα εναι να ξεχωριστ αντικεμενο (object). Στην ορολογα του OOP αυτο του εδου η σχση ονομζεται "**has-a relationship**". Α το δομε και στην πρξη:

```
class Piston():
    pass

class Valves():
    pass

class Cylinders():
    pass

class Engine():
    pistons = Pistons()
    valves = Valves()
    cylinders = Cylinders()
```

πω βλπουμε οι ιδιτητε (attributes) τη *Μηχαν* εναι *instances* μια λλη κλση. Στο παρδειγμα αυτ οι κλσει *Πιστονιο*, *Βαλβδα* και *Κυλνδρου* δεν χουν δικ του ιδιτητε/μεθδου, αυτ μω χει γνει καθαρ για λγου απλτητα. Δεν υπρχει καννα ττοιου τπου περιορισμ. Α δομε να ακμα παρδειγμα:

```
class Tire():
    size
    max_pressure

class Engine():
    cubic_capacity
    engine_type

class Car():
    model
    color

    tires = Tire(size=18, max_pressure=30)
    engine = Engine(type="V8", cubic_capacity=1800)
```

Λογικ πλον δεν χρειζονται πολλ εξηγσει. να αυτοκνητο χει ελαστικ και μηχαν. Ορζουμε αρχικ λοιπν τι κλσει *Ελαστικ* και *Μηχαν* με τι κατλληλε ιδιτητε και μεθδου. Στη συνχεια ορζουμε την κλση *Αυτοκνητο*

η οποα χει ω ιδιτητε (attributes) αντικεμενα Ελαστικο και αντικεμενα μηχαν. Πρα απ αυτ τι δο ιδιτητε, η κλση μπορε να χει και λλε ιδιτητε πω μοντλο και χρμα.

Α φτιξουμε τρα μερικ αντικεμενα Αυτοκιντου:

```
my_car = Car(model="Ford Escort", color="Blue")
your_car = Car(model="Ferrari Testarossa", color="Red")
```

Warning: Στο παρδειγμα αυτ, οι instances των ελαστικν και τη μηχαν δημιουργονται μσα στο σμα τη κλση Αυτοκνητο. Ω αποτλεσμα αυτο λα τα αντικεμενα Αυτοκιντου θα χουν την δια μηχαν και τα δια ελαστικ. Αν θλουμε να φτιξουμε αυτοκνητα με διαφορετικ μηχαν και ελαστικ πω θα το πετχουμε; Αυτ το ερτημα θα το απαντσουμε αργτερα, καθ εναι καλτερα να το εξηγσουμε χρησιμοποιντα την κανονικ σνταξη τη Python. Για την ρα λοιπν, δστε περισστερη προσοχ στην Ιδα τη Σνθεση και λιγτερο στην υλοποηση

Καλ λα αυτ... Εμνα γιατ μου χρειζονται;

Η απντηση σε αυτ το ερτημα δεν εναι και τσο απλ. Μια σντομη αναζτηση στο internet σχετικ με τα πλεονεκτματα του Αντικειμενοστραφο Προγραμματισμο (benefits/advantages of Object-Oriented Programming) θα επιστρψει πλθο σελδων που απαντον στο ερτημα αυτ. Οι απαντσει χωρζονται σε δο σκλη:

- Στα οφλη σε επεδο κδικα
- Στα οφλη σε επεδο οργνωση.

Τα οφλη τη πρτη κατηγορα εναι εκολο να γνουν αντιληπτ. Χρι στην κληρονομικτητα αποφεγουμε να επαναλβουμε κδικα (αρχ "Do not Repeat Yourself" - DRY). Αυτ γνεται γιατ λα τα αντικεμενα μα μοιρζονται τον διο κδικα. Οι κοιν μθοδοι ορζονται μα φορ στην υπερκλση και λε οι υποκλσει απλ την κληρονομον. Με τον τρπο αυτ μειντε το μγεθο του κδικα. Λιγτερο κδικα σημανει λιγτερα bugs και πιο κατανοητ κδικα, δηλαδ κδικα που εναι πιο εκολο να συντηρηθε.

Τα οφλη τη δετερη κατηγορα, η αλθεια εναι τι μπορε να τα εκτιμσει, μνο αφο χει δη κατανοσει και χρησιμοποισει τι αρχ του OOP στα προγρμματα σου. Εν πση περιπτσει προκειμνου να δσουμε μα απντηση, θα πρπει να πομε τι ο OOP εναι μα προσγγιση που μα επιτρπει να φτσουμε σε υψηλ επεδα αφαρεση (abstraction) μειοντα με αυτν τον τρπο την πολυπλοκτητα των προβλημτων που καλομαστε να λσουμε (και ποιο κατλαβε, κατλαβε :P).

CHAPTER 2

Indices and tables

- genindex
- search

p

P