

PYTHON – ΕΝΝΟΙΕΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΟΥΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

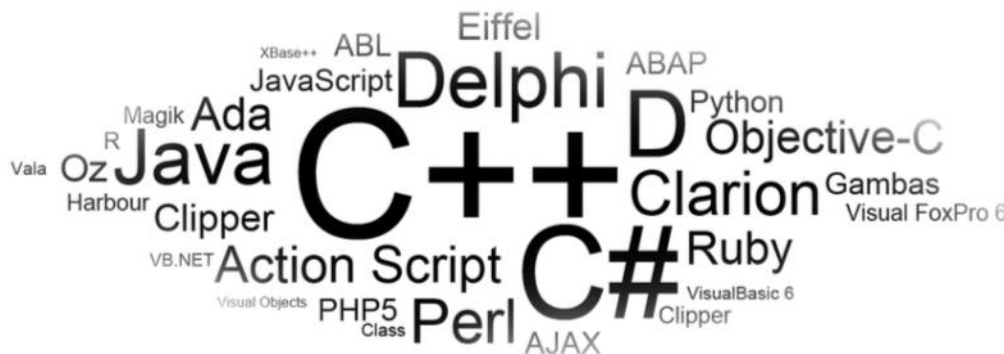


Αρχές Γλωσσών Προγραμματισμού και Μεταφραστών

Γ. Γαροφαλάκης, Σ. Σιούτας, Π. Χατζηδούκας

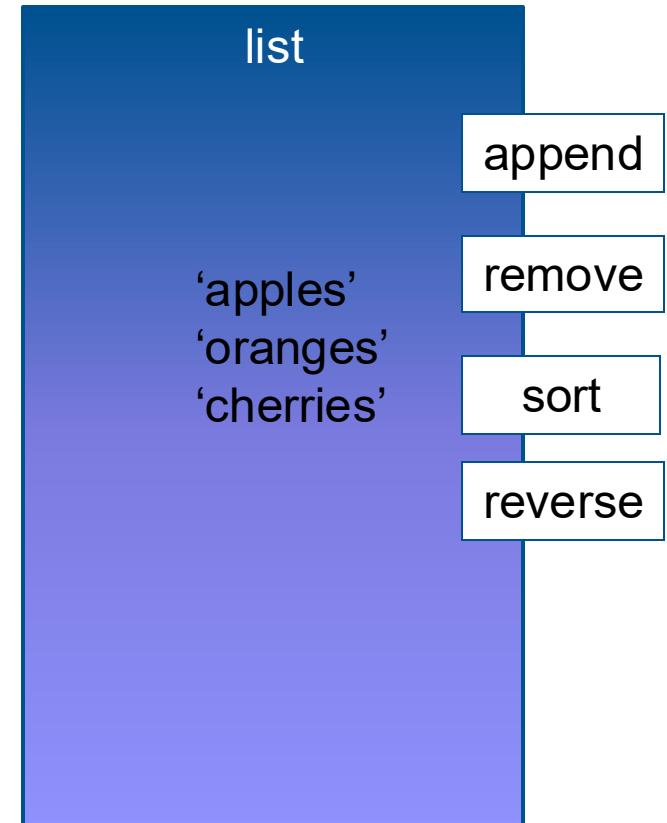
Αντικειμενοστρεφής Προγραμματισμός

- Ο Αντικειμενοστρεφής προγραμματισμός (Object Oriented Programming - OOP) είναι ένας τρόπος προγραμματισμού που υποστηρίζεται από πάρα πολλές και διαδεδομένες γλώσσες προγραμματισμού όπως τις:
 - Python
 - C#
 - C++
 - Java και πολλές ακόμα.



Αντικειμενοστρεφής Προγραμματισμός

- Ο αντικειμενοστρεφής προγραμματισμός έχει σαν πυρήνα τις **αφαιρέσεις**, γιατί **τα αντικείμενα είναι αφαιρέσεις** – οργανώνουν τη **λογική** και τα **δεδομένα** μας, **μειώνοντας** την πολυπλοκότητα.
- Αντί να έχουμε ένα γιγαντιαίο πρόγραμμα με κώδικα και μεταβλητές παντού, μπορούμε να έχουμε **αντικείμενα** που **ομαδοποιούν των κώδικα και τις μεταβλητές** μας.
- Η **λίστα** της Python:
 - Η λίστα είναι ένα αντικείμενο που είναι **δομή δεδομένων** – ένα δοχείο δεδομένων. Μπορούμε σε μία λίστα να αποθηκεύσουμε τα αντικείμενα ενός καλαθιού αγορών.
 - Επίσης, όμως εκθέτει **μεθόδους** για να διαχειριστούμε αυτά τα δεδομένα.



Αντικειμενοστρεφής Προγραμματισμός

- Ένας λοιπόν από τους **στόχους του OOP** είναι να πάρεις ότι είναι απαραίτητο σε σχέση με μία εφαρμογή και **να βρεις τρόπο να δημιουργήσεις αντικείμενα που να ομαδοποιήσουν τον κώδικα και τα δεδομένα που χρειάζεσαι**. Αυτό μειώνει την πολυπλοκότητα.
- Για να φτιάξουμε όμως το πρόγραμμά μας, εκτός από τα έτοιμα αντικείμενα που μας δίνει η Python χρειαζόμαστε και **αντικείμενα που θα ορίσουμε εμείς** και αναπαριστούν τις **ανάγκες της εφαρμογής μας**.
- Τα αντικείμενα που θα **δημιουργήσουμε** μπορεί να αναπαριστούν **αντικείμενα του πραγματικού κόσμου**, αλλά αυτό δεν είναι και απαραίτητο.



Αντικείμενα (object) (1/2)

- Προκειμένου να εξηγήσουμε τον όρο αντικείμενο (object) βοηθάει να σκεφτόμαστε τα αντικείμενα (objects) ως κάτι ανάλογο με αυτό που ονομάζουμε στη γραμματική της φυσικής γλώσσας “ουσιαστικό”.

*«Πρόκειται για Οντότητες (έμψυχες ή άψυχες) οι οποίες έχουν συγκεκριμένες **ιδιότητες** ή/και μπορούν να **εκτελούν συγκεκριμένες ενέργειες**»*

Αντικείμενα (object) (2/2)

Ας δούμε μερικά παραδείγματα:

- Μία καρέκλα έχει τις ιδιότητες **χρώμα**, **ύψος**, **υλικό** κτλ.
- Ένα ορθογώνιο αντίστοιχα έχει τις ιδιότητες **πλάτος**, **ύψος**, **εμβαδόν**, **περίμετρος** κτλ.
- Μία γάτα έχει τις ιδιότητες **όνομα**, **ηλικία**, **φύλλο** κτλ αλλά επίσης μπορεί και να εκτελεί διάφορες ενέργειες όπως π.χ. να **νιαουρίσει**, να **τρέξει**, να **περπατήσει**, να **σκαρφαλώσει**, να **φάει** κτλ.
- Ένας οδηγός μπορεί να προβεί σε διάφορες ενέργειες, όπως πχ να **στρίψει**, να **φρενάρει**, να **επιταχύνει**, να **βάλει μπροστά τη μηχανή** κτλ.

Ας φτιάξουμε λοιπόν ένα δικό μας αντικείμενο...

- Όταν δημιουργούμε ένα αντικείμενο θα πρέπει να υπάρχει ένα **σχεδιάγραμμα** (blueprint) για αυτό, ορισμένο κάπου.
- Η **κλάση** είναι αυτό το σχεδιάγραμμα.
- Ας φτιάξουμε ένα αντικείμενο **Person**.
- Μία **μέθοδος** είναι μία **συνάρτηση** που έχει συσχετιστεί με ένα αντικείμενο.

Μία **σύμβαση** που χρησιμοποιούμε στην Python είναι το πρώτο γράμμα της κλάσης να είναι κεφαλαίο.

```
class Person:
    def say_hello(self):
        print("Hello!")

p1 = Person()
p1.say_hello()
```



```
>>> ===== RESTART =====
>>> Hello!
>>>
```

Δεν μπορούμε να χρησιμοποιήσουμε τον ορισμό της κλάσης σαν αντικείμενο, αλλά μπορούμε να δημιουργήσουμε αντικείμενα από την κλάση. Αυτή η διαδικασία λέγεται **Instantiation**.

Δεν μπορώ να γράψω **Person.say_hello()** .

Κλάσεις (Classes)

- Μια κλάση δεν είναι τίποτα άλλο παρά ένας **ορισμός**. Αυτό που ορίζει είναι το ποιες ιδιότητες (attributes) και ποιες μεθόδους (methods) θα έχει ένα αντικείμενο.
- Στην ορολογία του Αντικειμενοστρεφούς Προγραμματισμού, όλα τα νέα αντικείμενα που δημιουργούνται από μία κλάση ονομάζονται *στιγμιότυπα* (instances).

```
class Cat():  
    name  
    age  
    sex  
    def eat():  
        print("%s is eating" % name)  
    def sleep():  
        print("%s is sleeping" %  
name)  
  
my_cat = Cat(name="Kitty",  
age=2, sex="female")  
your_cat = Cat(name="Paul",  
age=8, sex="male")
```

Κληρονομικότητα (Inheritance) (1/3)

- Ίσως η πλέον θεμελιώδης έννοια του Αντικειμενικοστραφούς Προγραμματισμού.
- Με τον όρο κληρονομικότητα, εννοούμε τη δυνατότητα που έχει μια κλάση να *κληρονομεί* όλες τις ιδιότητες και τις μεθόδους μιας άλλης κλάσης.

Κληρονομικότητα (Inheritance) (2/3)

Κλάση οποία δημιουργεί Ζώα:

```
class Animal():  
    species  
    race  
    sex  
  
    def speak():  
        print("I don't know what to  
say...")
```

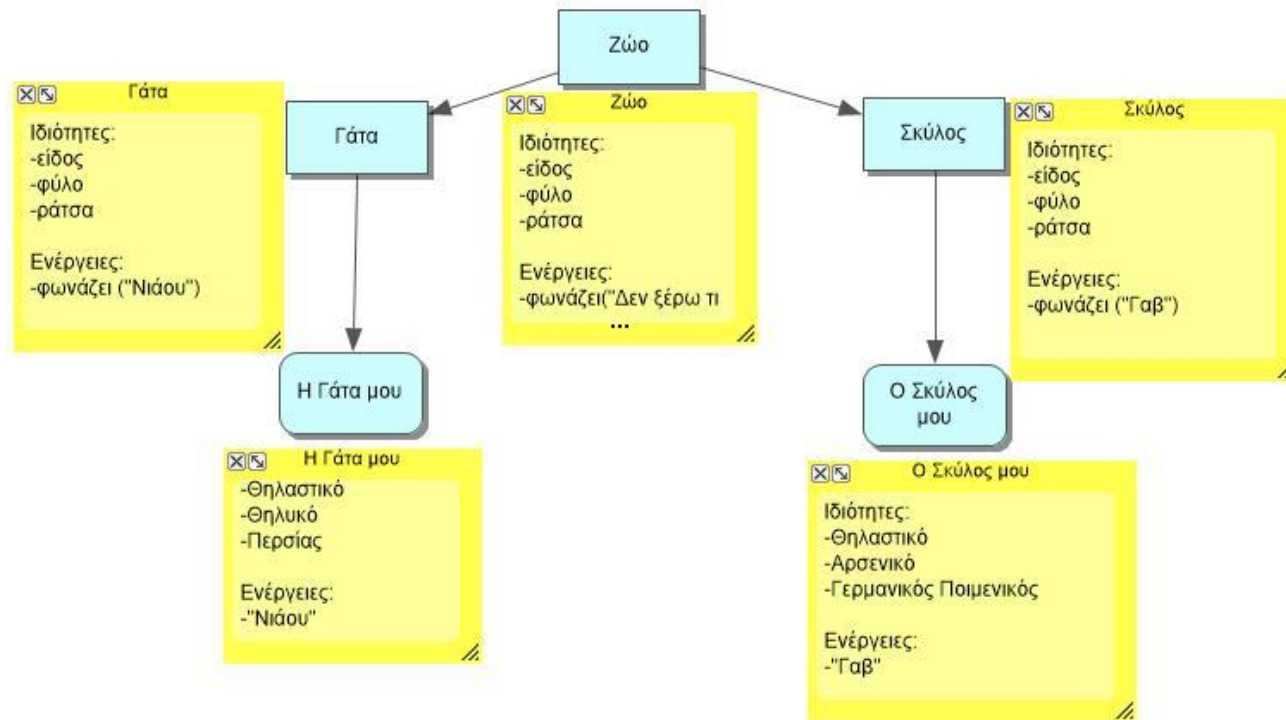
Υποκλάσεις που κληρονομούν την κλάση Animal

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

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

Οι κλάσεις Σκύλου και Γατας κληρονομούν τις ιδιότητες και τις μεθόδους της κλάσης Ζώου. Προσοχή όμως στη μέθοδο `speak()`! Όπως βλέπουμε και οι δύο υποκλάσεις ξαναορίζουν την μέθοδο που κληρονόμησαν από την υπερκλάση τους. Αυτό είναι μία από τις βασικότερες τεχνικές της Κληρονομικότητας (Inheritance). Οι υποκλάσεις δεν κληρονομούν απλά ιδιότητες (attributes) και μεθόδους (methods), αλλά μπορούν να ξαναορίσουν τις μεθόδους που κληρονόμησαν, εξειδικεύοντάς τις.

Κληρονομικότητα (Inheritance) (3/3)



Instantiation (1/2)

- Μπορούμε να κάνουμε **instantiate** πολλαπλά αντικείμενα από τον ίδιο ορισμό κλάσης και αυτό είναι πολύ χρήσιμο.

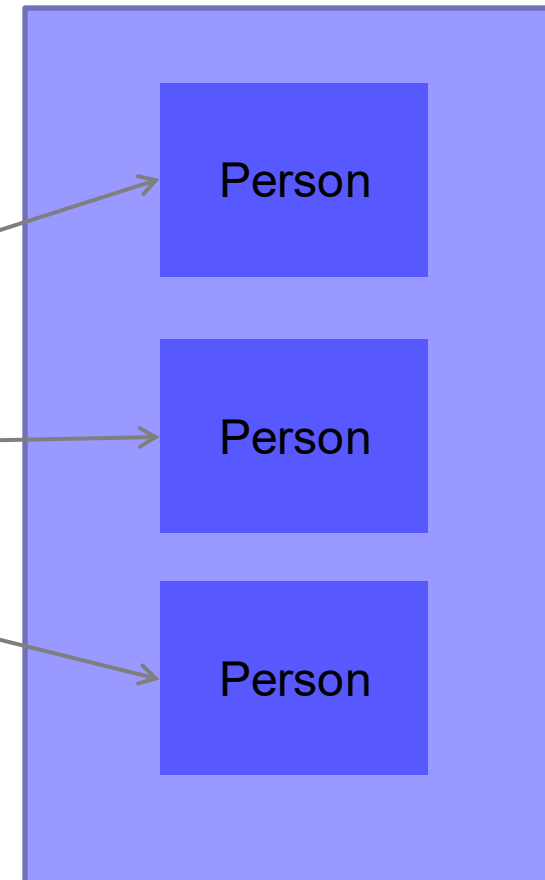
- ✦ Για να γράψουμε ένα μεγάλο πρόγραμμα σίγουρα χρειαζόμαστε πολλαπλά αντικείμενα.

- ✦ Για την περίπτωση μας τα αντικείμενα **Person** θα αναπαριστούν διαφορετικούς ανθρώπους.

- ✦ Κάθε ένα από αυτά τα αντικείμενα θα δημιουργείται διακριτά στη **μνήμη** και θα κρατάει διαφορετικά τμήματα δεδομένων. Αυτό το ονομάζουμε **κατάσταση / state** του αντικειμένου.

```
p1 = Person()  
p2 = Person()  
p3 = Person()
```

- ✦ Η **κατάσταση** ενός αντικειμένου αποτελείται από τα **δεδομένα** που κρατάει.



Instantiation (2/2)

- Προς το παρόν δεν έχουμε δεδομένα σε κάθε αντικείμενο και ας δούμε πως γίνεται αυτό.
- Όλες οι γλώσσες που επιτρέπουν τη δημιουργία αντικειμένων από κλάσεις, έχουν κάποια **ειδική μέθοδο** για την **αρχικοποίηση των αντικειμένων** με συγκεκριμένα κομμάτια δεδομένων.
- Κάποιες γλώσσες αποκαλούν αυτές τις μεθόδους **constructors** / **κατασκευαστές**, γιατί χρησιμοποιούνται για την κατασκευή ενός αντικειμένου.
- Στην Python αυτή η μέθοδος είναι η **init** (από το initialize).

```
class Person:

    def __init__(self):
        self.name = "John"

    def say_hello(self):
        print("Hello!", self.name)

p1 = Person()
p1.say_hello()
```



```
>>> =====
>>>
Hello! John
>>>
```

Η μέθοδος init

- Αυτός ο τρόπος γραφής είναι μία **σύμβαση** για μερικές από τις ειδικές μεθόδους της Python, με τις οποίες **δεν αλληλεπιδρούμε**.
Δε θα γράφαμε ποτέ **p1.__init__**

- Αυτό **το κάνει η Python για εμάς** όταν κατασκευάζουμε ένα αντικείμενο.
- Όταν κάνουμε instantiate ένα αντικείμενο το **πρώτο πράγμα** που κάνει η Python είναι **να εκτελέσει την init μέθοδο για εμάς**. Για αυτό το λόγο είναι ο **initializer** (constructor). Μου επιτρέπει φτιάξω κάποια πράγματα εντός του αντικειμένου πριν εκτελεστεί ο υπόλοιπος κώδικας.
- Η παράμετρος που περνάμε στην **__init__** δε χρειάζεται να είναι η **self**, αλλά αυτό είναι μία ακόμα σύμβαση.

```
class Person:
    def __init__(self):
        self.name = "John"

    def say_hello(self):
        print("Hello!", self.name)

p1 = Person()
p1.say_hello()
```

Δεν χρειάζεται να περάσουμε κάποια παράμετρο όταν καλούμε κάποια μέθοδο του αντικειμένου και αυτό γιατί η ίδια η Python θα περάσει μία παράμετρο η οποία ουσιαστικά θα είναι μία αναφορά στο αντικείμενο όπου αυτή η μέθοδος καλείται.

Attributes - Κατηγορήματα

- Τα **attributes** / **κατηγορήματα** είναι στην πραγματικότητα τα τμήματα των δεδομένων τα οποία το αντικείμενο εκθέτει. Σε άλλες γλώσσες ονομάζονται **properties**.
- Και αν θέλουμε να το κάνουμε λίγο πιο ενδιαφέρον, ώστε όλα τα αντικείμενα να μην έχουν το ίδιο **name**.

```
class Person:

    def __init__(self):
        self.name = "John"

    def say_hello(self):
        print("Hello!", self.name)

p1 = Person()
p1.say_hello()
```

```
class Person:

    def __init__(self):
        self.name = "John"

    def say_hello(self):
        print("Hello, ", self.name)

p1 = Person()
p1.name = "Maria"
p1.say_hello()

p2 = Person()
p2.say_hello()
```

```
>>>
Hello,  Maria
Hello,  John
>>>
```

```
class Person:

    def __init__(self, name):
        self.name = name

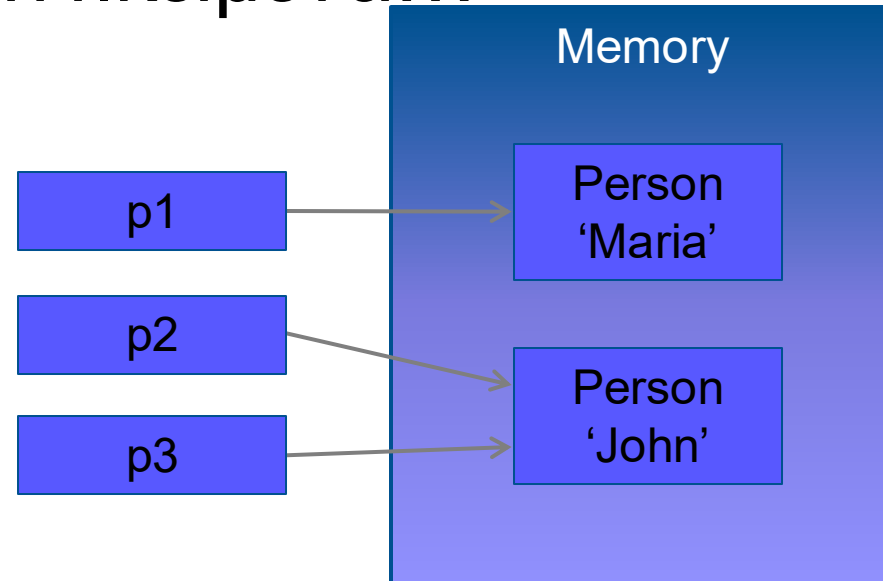
    def say_hello(self):
        print("Hello, ", self.name)

p1 = Person("Maria")
p1.say_hello()

p2 = Person("John")
p2.say_hello()
```

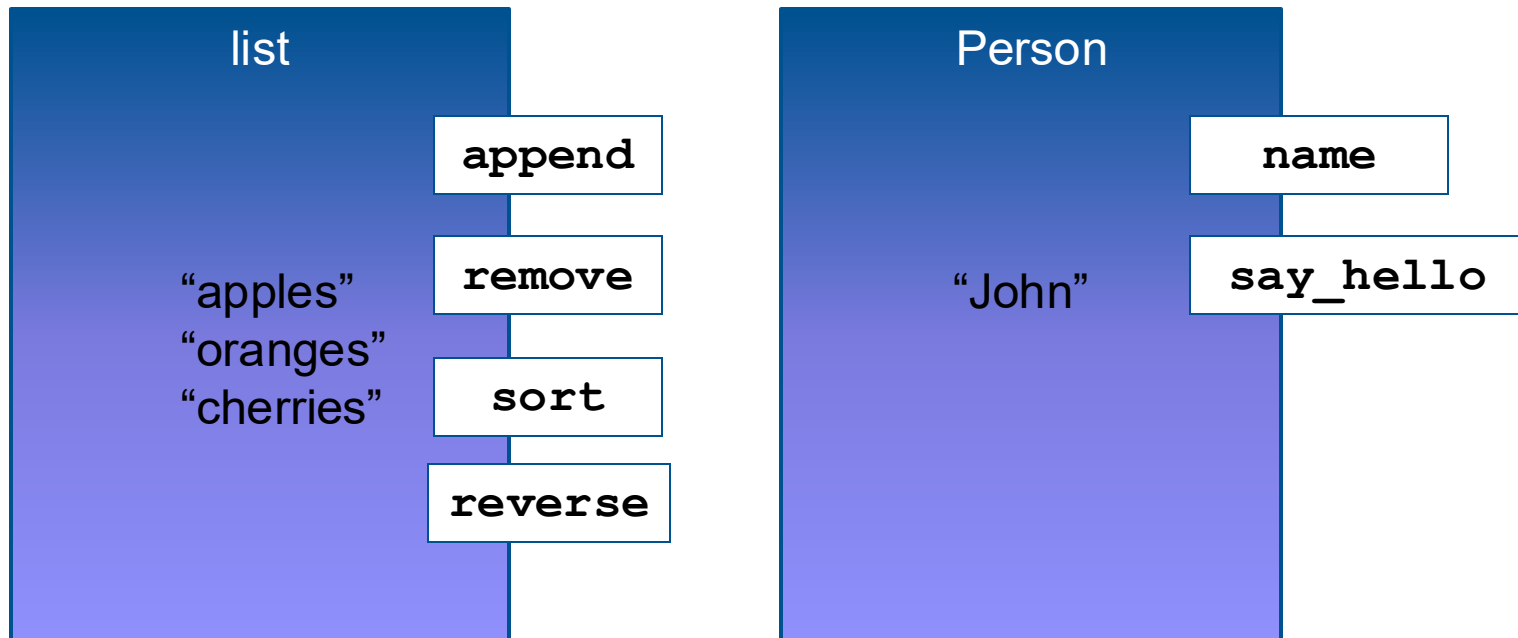
Και ας δούμε τώρα τη συσχέτιση των μεταβλητών με τα αντικείμενα...

- Όταν δημιουργούμε ένα αντικείμενο αυτό καταλαμβάνει κάποιο χώρο στη μνήμη του υπολογιστή.
- Οι μεταβλητές δεν περιέχουν τα αντικείμενα. Τις χρησιμοποιούμε για να έχουμε πρόσβαση σε αυτά.
- Τις μεταβλητές μπορούμε να τις σκεφτόμαστε ως δείκτες ή αναφορές.
- Περιέχουν τη θέση που μπορούμε να βρούμε ένα αντικείμενο.
- Μία μεταβλητή μπορεί να δείχνει ένα αντικείμενο κάθε φορά, αλλά δύο διαφορετικές μεταβλητές μπορούν να δείχνουν στο ίδιο αντικείμενο.

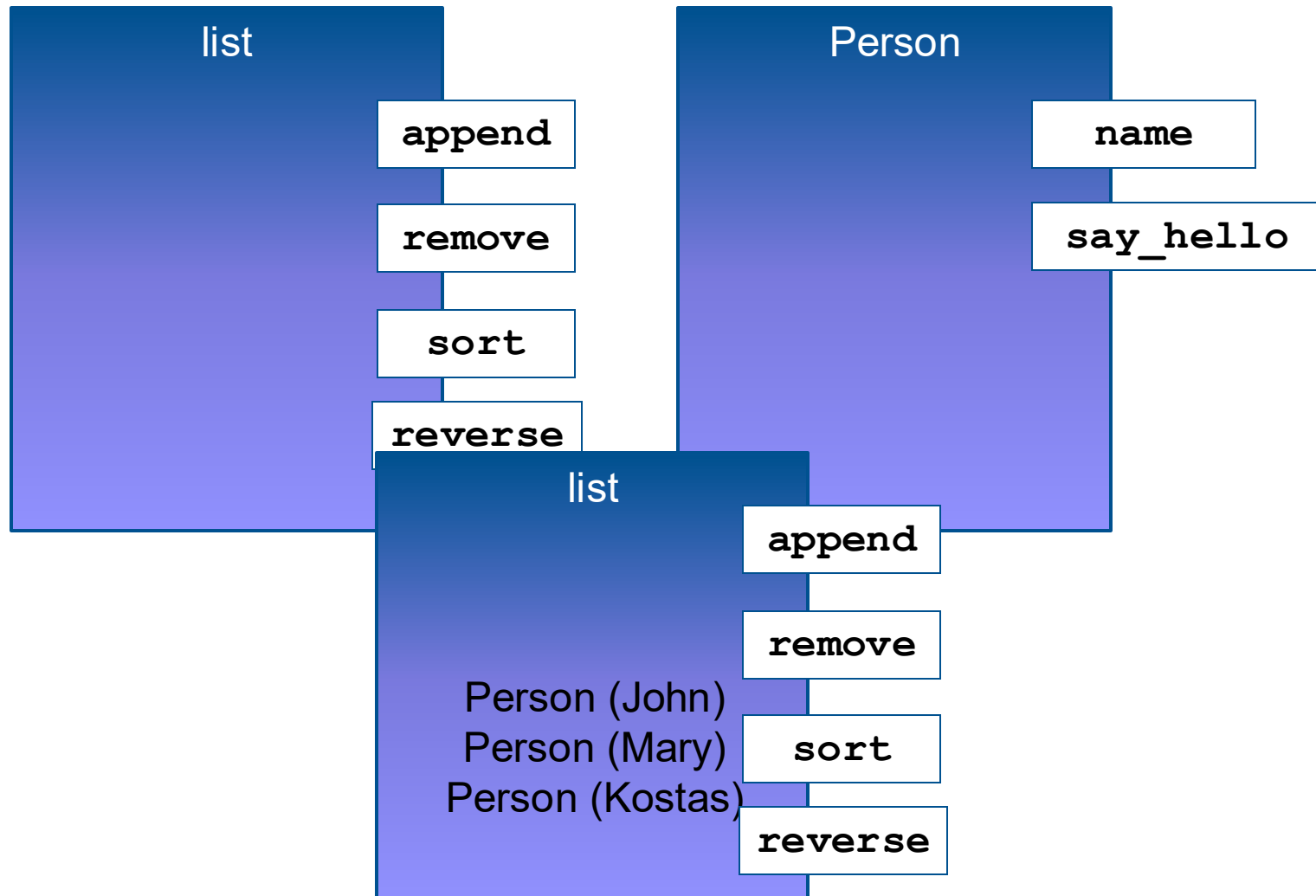


Ενθυλάκωση – Encapsulation

- Οι αντικειμενοστρεφείς τεχνικές είναι δημοφιλείς τα τελευταία 20 χρόνια γιατί μας επιτρέπουν να δημιουργήσουμε **αφαιρέσεις** (γενικεύουν και απλοποιούν τον κώδικά μας μειώνοντας την πολυπλοκότητά του).
- Ένας τρόπος για να γίνει αυτό είναι η **ενθυλάκωση**. Υπάρχουν δύο τρόποι για να σκέφτεστε την ενθυλάκωση:
 - **Απόκρυψη λεπτομερειών** (ξέρετε πως ακριβώς λειτουργεί ένα list ή πως ακριβώς λειτουργεί ένα αυτοκίνητο πριν το οδηγήσετε;)
 - **Ομαδοποίηση λεπτομερειών σε ένα συνεκτικό μοντέλο: Κλάσεις** (Ενσωματωμένα δεδομένα και συμπεριφορά).



Σύνθεση – Composition



- Το πώς **συνθέτουμε** όλα μαζί τα διαφορετικά αντικείμενα για να κάνουμε κάτι μεγαλύτερο – μία πιο στοχευμένη αφαίρεση.

Ας συνθέσουμε...

- Πως διαχειριζόμαστε μία τάξη?
- Μία τάξη μπορεί να περιέχει πολλά άτομα.
- Θα πρέπει να φτιάξουμε μία αφαίρεση για να διαχειριζόμαστε πολλαπλά αντικείμενα τύπου **Person** και να τα ομαδοποιούμε σε μία τάξη.



...στην πράξη...

Σύμβαση: Όταν ονοματίζετε ένα attribute με _ σαν πρώτο γράμμα, τότε θα χρησιμοποιείται μόνο εντός της κλάσης. Είναι μία μορφή ενθυλάκωσης.

```
class Classroom:

    def __init__(self):
        self._people = {}

    def add_person(self, person):
        self._people.append(person)

    def remove_person(self, person):
        self._people.remove(person)

    def greet(self):
        for person in self._people:
            person.say_hello()

class Person:

    def __init__(self, name):
        self.name = name

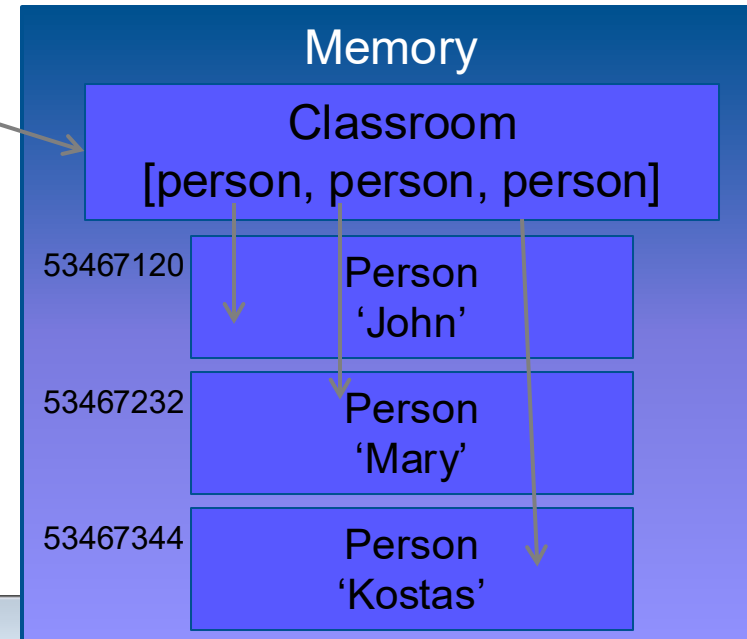
    def say_hello(self):
        print(id(self))
        print("Hello, ", self.name)

room = Classroom()
room.add_person(Person("John"))
room.add_person(Person("Mary"))
room.add_person(Person("Kostas"))

room.greet()
```



```
Python 3.3.3 Shell
File Edit Shell Debug Options Windows Help
Type "copyright", "credits" or "license()" fo
>>> ===== RESTART :
>>>
53467120
Hello, John
53467232
Hello, Mary
53467344
Hello, Kostas
>>>
```



```

class Classroom:

    def __init__(self):
        self._people = []

    def add_person(self, person):
        self._people.append(person)

    def remove_person(self, person):
        self._people.remove(person)

    def greet(self):
        for person in self._people:
            person.say_hello()

class Person:

    def __init__(self, name):
        self.name = name

    def say_hello(self):
        print(id(self))
        print("Hello, ", self.name)

room = Classroom()
room.add_person(Person("John"))
room.add_person(Person("Mary"))
room.add_person(Person("Kostas"))

room.greet()

```

Κλάση Classroom:

Αντιπροσωπεύει μια "τάξη" που μπορεί να περιέχει άτομα.

Μέθοδοι:

`__init__(self)`

Δημιουργεί ένα νέο αντικείμενο Classroom. Αρχικοποιεί μια λίστα `self._people` που θα περιέχει αντικείμενα τύπου Person.

`add_person(self, person)`

Προσθέτει ένα άτομο (αντικείμενο Person) στη λίστα `self._people`.

`remove_person(self, person)`

Αφαιρεί το συγκεκριμένο άτομο από τη λίστα `self._people`.

`greet(self)`

Καλεί τη μέθοδο `say_hello()` για κάθε άτομο στην τάξη.

Κλάση Person:

Αντιπροσωπεύει ένα άτομο με όνομα.

Μέθοδοι:

`__init__(self, name)`

Ο constructor που αποθηκεύει το όνομα του ατόμου στο `self.name`.

`say_hello(self)`

Εκτυπώνει:

- το αναγνωριστικό αντικειμένου με `id(self)`
- και ένα μήνυμα καλωσορίσματος π.χ. "Hello, John"

- Οι κλάσεις της Python που έχουν **attributes** και **methods**.
- Μας βοηθάνε να δημιουργήσουμε **αφαιρέσεις**.
- Σημαντικές επίσης οι λειτουργίες της **ενθυλάκωσης** και της **σύνθεσης**.



```

def get_order():
    print("[command] [item] (command is a to add, d to delete, q to quit)")
    line = input()

    command = line[:1]
    item = line[2:]

    return command, item

def add_to_cart(item, cart):
    if not item in cart:
        cart[item] = 0
    cart[item] += 1

def delete_from_cart(item, cart):
    if item in cart:
        cart[item] -= 1

def process_order(order, cart):
    command, item = order

    if command == "a":
        add_to_cart(item, cart)
    elif command == "d" and item in cart:
        delete_from_cart(item, cart)
    elif command == "q":
        return False

    return True

def go_shopping():
    cart = dict()

    while True:
        order = get_order()
        if not process_order(order, cart):
            break

    print(cart)
    print("Finished!")

```

- Θα δημιουργήσουμε ένα εικονικό καλάθι αγορών.
- Έχουμε χρησιμοποιήσει μόνο **συναρτήσεις**. Και με αυτές δημιουργούμε **αφαιρέσεις**.
- Ο κώδικας έχει δυο βασικές έννοιες:
 - Την παραγγελία – **order**
 - Το καλάθι αγορών – **cart**

```

def get_order():
    print("[command] [item] (command is a to add, d to delete, q to quit)")
    line = input()

    command = line[:1]
    item = line[2:]

    return command, item

def add_to_cart(item, cart):
    if not item in cart:
        cart[item] = 0
    cart[item] += 1

def delete_from_cart(item, cart):
    if item in cart:
        cart[item] -= 1

def process_order(order, cart):
    command, item = order

    if command == "a":
        add_to_cart(item, cart)
    elif command == "d" and item in cart:
        delete_from_cart(item, cart)
    elif command == "q":
        return False

    return True

def go_shopping():
    cart = dict()

    while True:
        order = get_order()
        if not process_order(order, cart):
            break

    print(cart)
    print("Finished!")

```

1. Συνάρτηση get_order()

-Ζητάει είσοδο από τον χρήστη σε μορφή π.χ. "a apple" ή "d banana".

-Διαχωρίζει την εντολή (π.χ. a, d, q) από το αντικείμενο (item).

-Επιστρέφει μια πλειάδα (command, item).

2. Συνάρτηση add_to_cart(item, cart)

Αν το αντικείμενο δεν υπάρχει στο καλάθι (cart), το προσθέτει με τιμή 0.

Αυξάνει κατά 1 την ποσότητα του συγκεκριμένου αντικειμένου.

3. Συνάρτηση delete_from_cart(item, cart)

Αν το αντικείμενο υπάρχει στο καλάθι, μειώνει την ποσότητά του κατά 1.

4. Συνάρτηση process_order(order, cart)

Διαχειρίζεται τις εντολές:

- "a" → προσθήκη στο καλάθι
- "d" → αφαίρεση από το καλάθι (αν υπάρχει)
- "q" → σταματάει το πρόγραμμα

5. Συνάρτηση go_shopping()

Ξεκινά με ένα άδειο καλάθι.

Επαναλαμβάνει:

- Διαβάζει εντολή από τον χρήστη.
- Την επεξεργάζεται.
- Αν η εντολή είναι q, σταματάει τη διαδικασία.
- Εκτυπώνει το περιεχόμενο του καλαθιού και το μήνυμα "Finished!".

Τι πρέπει να σκεφτόμαστε όμως για να επιλέξουμε τις κλάσεις μας;

- **Αρχή #1:** Να δίνετε στις κλάσεις σας μία **μοναδική ευθύνη**. Για το καλάθι μας η ευθύνη αυτή ίσως είναι απλά να αποθηκεύει τα αντικείμενα που έχει επιλέξει ο χρήστης. Οι κλάσεις σας πρέπει να είναι **μικρές** και **στοχευμένες**.
- **Αρχή #2:** Να **σχεδιάζετε** τις κλάσεις σας **από έξω**. Γράψτε πρώτα τον κώδικα που θα χρησιμοποιήσει την κλάση πριν από τον κώδικα της κλάσης.

```
class Cart:

    def __init__(self):
        self._contents = dict()

class Order:
```

```
cart = Cart()
order = Order()
order.get_input()

while not order.quit:

    order = Order()
    order.get_input()
```

Φτιάχνουμε τις κλάσεις μας:

```
class Cart:

    def __init__(self):
        self._contents = dict()

class Order:

    def __init__(self):
        self.quit = False

    def get_input(self):
        print("[command] [item] (command is a to add, d to delete, q to quit)")
        line = input()

        command = line[:1]
        item = line[2:]

        if command == "q":
            self.quit = True

cart = Cart()
order = Order()
order.get_input()

while not order.quit:

    order = Order()
    order.get_input()
```

```
>>> ===== RESTART =====
>>>
[command] [item] (command is a to add, d to delete, q to quit)
q
>>>
```

Φτιάχνουμε τις κλάσεις μας:

```
class Cart:
```

```
    def __init__(self):  
        self._contents = dict()
```

```
class Order:
```

```
    def __init__(self):  
        self.quit = False
```

```
    def get_input(self):  
        print("[command] [item] (command is a to add, d to delete, q to quit)")  
        line = input()
```

```
        command = line[:1]  
        item = line[2:]
```

```
        if command == "q":  
            self.quit = True
```

```
cart = Cart()  
order = Order()  
order.get_input()
```

```
while not order.quit:
```

```
    order = Order()  
    order.get_input()
```

Κλάση Cart

-Δημιουργεί ένα αντικείμενο "καλάθι".

-Η μεταβλητή _contents είναι ένα λεξικό (dict) όπου θα αποθηκεύονται τα προϊόντα με τις αντίστοιχες ποσότητες.

Κλάση Order

Αρχειοποιεί την ιδιότητα quit ως False, για να δηλώνει αν ο χρήστης θέλει να σταματήσει τις αγορές.

Μέθοδος get_input

Εκτυπώνει τις οδηγίες για τον χρήστη.
Λαμβάνει είσοδο από τον χρήστη (π.χ. "a apple", "q").
Χωρίζει την εντολή (command) και το προϊόν (item).
Αν η εντολή είναι "q", τότε η ιδιότητα quit γίνεται True, δηλώνοντας ότι ο χρήστης θέλει να τερματίσει.

Κύριο πρόγραμμα (main)

-Δημιουργείται ένα καλάθι (cart = Cart()).

-Δημιουργείται μια παραγγελία (order = Order()).

-Καλείται η get_input() για να διαβαστεί η πρώτη εντολή του χρήστη.

-Όσο η εντολή δεν είναι "q" (δηλ. δεν έχει τεθεί order.quit = True), επαναλαμβάνεται:

-Δημιουργείται νέο αντικείμενο Order.

-Ζητείται νέα είσοδος από τον χρήστη.

```
>>> ===== RESTART =====  
>>>  
[command] [item] (command is a to add, d to delete, q to quit)  
q  
>>>
```

```
class Cart:
```

```
def __init__(self):
    self._contents = dict()

def process(self, order):
    if order.add:
        if not order.item in self._contents:
            self._contents[order.item] = 0
        self._contents[order.item] += 1
    elif order.delete:
        if order.item in self._contents:
            self._contents[order.item] -= 1
            if self._contents[order.item] <= 0:
                del self._contents[order.item]
```

```
class Order:
```

```
def __init__(self):
    self.quit = False
    self.add = False
    self.delete = False
    self.item = None

def get_input(self):
    print("[command] [item] (command is a to add, d to delete, q to quit)")
    line = input()

    command = line[:1]
    self.item = line[2:]

    if command == "a":
        self.add = True
    elif command == "d":
        self.delete == True
    elif command == "q":
        self.quit = True
```

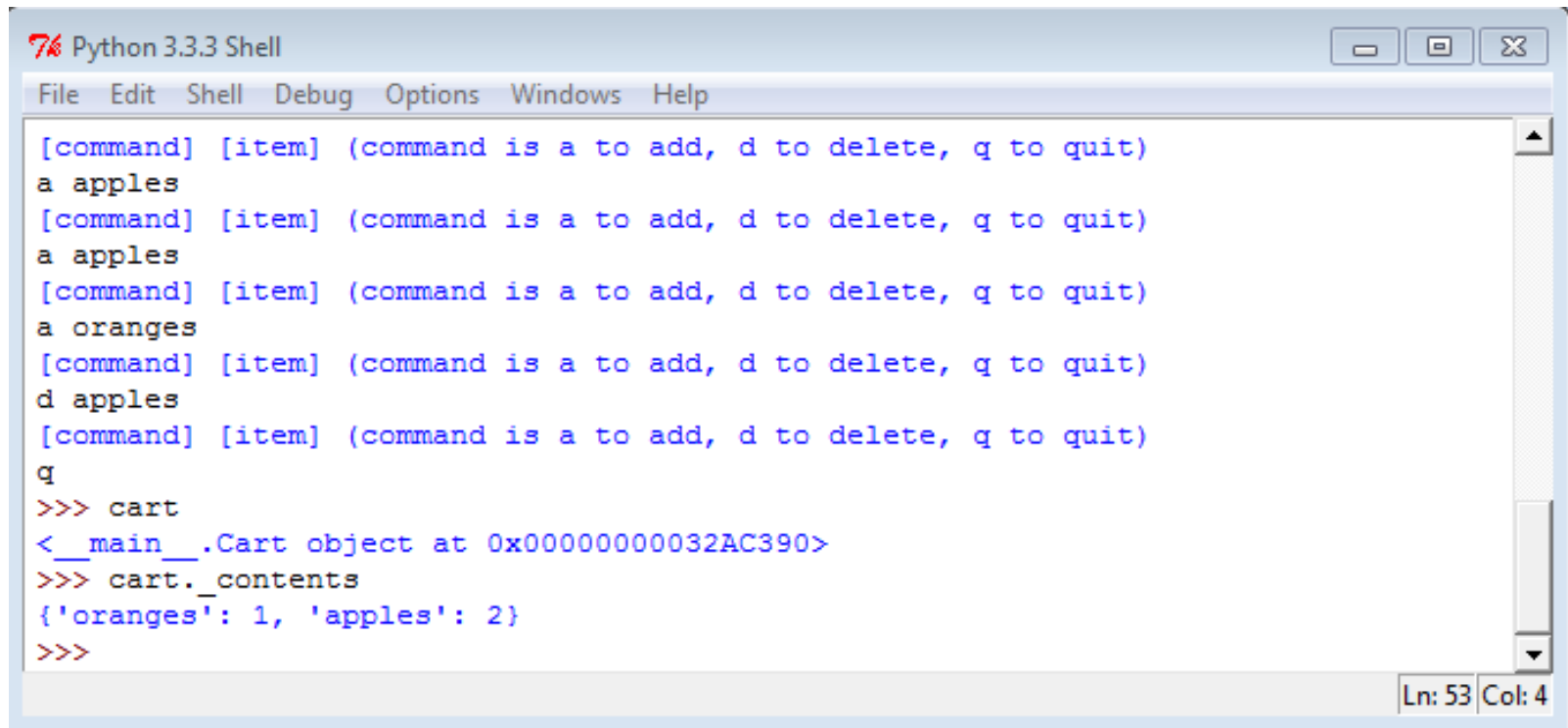
Κυρίως πρόγραμμα

```
cart = Cart()
order = Order()
order.get_input()

while not order.quit:
    ➡ cart.process(order)
    order = Order()
    order.get_input()
```

αφήνουμε αυτή τη διαδικασία στο Order.

Αυτό που έχει το καλάθι μας μέχρι στιγμής



```
Python 3.3.3 Shell
File Edit Shell Debug Options Windows Help

[command] [item] (command is a to add, d to delete, q to quit)
a apples
[command] [item] (command is a to add, d to delete, q to quit)
a apples
[command] [item] (command is a to add, d to delete, q to quit)
a oranges
[command] [item] (command is a to add, d to delete, q to quit)
d apples
[command] [item] (command is a to add, d to delete, q to quit)
q
>>> cart
<__main__.Cart object at 0x00000000032AC390>
>>> cart._contents
{'oranges': 1, 'apples': 2}
>>>
```

Ln: 53 Col: 4

```

>>> ===== RESTART =====
>>>
[command] [item] (command is a to add, d to delete, q to quit)
a apples
[command] [item] (command is a to add, d to delete, q to quit)
a apples
[command] [item] (command is a to add, d to delete, q to quit)
a oranges
[command] [item] (command is a to add, d to delete, q to quit)
d apples
[command] [item] (command is a to add, d to delete, q to quit)
q
>>> cart
<class '__main__.Cart'> {'_contents': {'apples': 2, 'oranges': 1}}
>>> cart._contents
{'apples': 2, 'oranges': 1}
>>> Cart
<class '__main__.Cart'>
>>> cart.__dict__
{'_contents': {'apples': 2, 'oranges': 1}}
>>> order.__dict__
{'add': False, 'item': '', 'delete': False, 'quit': True}
>>> order.__dict__["quit"]
True
>>> ===== RESTART =====
>>>
[command] [item] (command is a to add, d to delete, q to quit)
a apples
[command] [item] (command is a to add, d to delete, q to quit)
a oranges
[command] [item] (command is a to add, d to delete, q to quit)
q
<class '__main__.Cart'> {'_contents': {'oranges': 1, 'apples': 1}}
>>>

```

```

class Cart:

    def __init__(self):
        self._contents = dict()

    def __repr__(self):
        return "{0} {1}".format(Cart, self.__dict__)

```

Ας δούμε τι περιέχει
το καλάθι μας

- Υπάρχουν ειδικές **methods** και **attributes** που μας βοηθάνε.

Κυρίως πρόγραμμα

```

cart = Cart()
order = Order()
order.get_input()

while not order.quit:
    cart.process(order)
    order = Order()
    order.get_input()

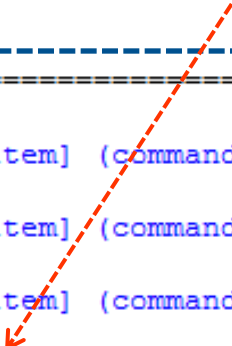
print(cart)

```

Python Modules

- Σχεδόν κάθε προγραμματιστικό περιβάλλον παρέχει τρόπο για **ομαδοποίηση** και **οργάνωση** κώδικα.
- Το έχουμε ήδη κάνει με τις **συναρτήσεις** και τις **κλάσεις**.
- Μπορούμε όμως να πετύχουμε ακόμα παραπάνω ομαδοποιώντας τον κώδικά μας σε **διαφορετικά αρχεία**.
- Στην περίπτωση της Python αυτό γίνεται με τα **modules** και ήδη έχουμε χρησιμοποιήσει ένα module.

```
>>> ===== RESTART =====
>>>
[command] [item] (command is a to add, d to delete, q to quit)
a apples
[command] [item] (command is a to add, d to delete, q to quit)
a oranges
[command] [item] (command is a to add, d to delete, q to quit)
q
<class '__main__.Cart'> {'_contents': {'oranges': 1, 'apples': 1}}
>>> __name__
'__main__'
>>>
```



```
class Cart:

    def __init__(self):
        self._contents = dict()

    def __repr__(self):
        return "{0} {1}".format(Cart, self.__dict__)

    def process(self, order):
        if order.add:
            if not order.item in self._contents:
                self._contents[order.item] = 0
            self._contents[order.item] += 1
        elif order.delete:
            if order.item in self._contents:
                self._contents[order.item] -= 1
                if self._contents[order.item] <= 0:
                    del self._contents[order.item]
```

```
class Order:

    def __init__(self):
        self.quit = False
        self.add = False
        self.delete = False
        self.item = None

    def get_input(self):
        print("[command] [item] (command is a to add, d to delete, q to quit)")
        line = input()

        command = line[:1]
        self.item = line[2:]

        if command == "a":
            self.add = True
        elif command == "d":
            self.delete = True
        elif command == "q":
            self.quit = True
```

Sales.py

Κυρίως πρόγραμμα

```
import sales

cart = sales.Cart()
order = sales.Order()
order.get_input()

while not order.quit:
    cart.process(order)
    order = sales.Order()
    order.get_input()

print(cart)
```

Προσέξτε το module το οποίο έχουμε.

```
>>> ===== RESTART =====
>>>
[command] [item] (command is a to add, d to delete, q to quit)
a apples
[command] [item] (command is a to add, d to delete, q to quit)
a oranges
[command] [item] (command is a to add, d to delete, q to quit)
q
<class 'sales.Cart'> {'_contents': {'apples': 1, 'oranges': 1}}
>>>
```

Python Packages

- Όταν έχουμε κώδικα σε **πολλαπλούς καταλόγους**.
- Στην Python ένας υποκατάλογος γίνεται **package**.
- Όπως οι κλάσεις έχουν **init** μέθοδο, έτσι μπορεί να έχει και ένα **module** ή ένα **package**. Για το λόγο αυτό φτιάχνουμε ένα κενό αρχείο στον ίδιο φάκελο με το όνομα **__init__.py**



Κυρίως πρόγραμμα

```
import sales.shopping_cart, sales.shopping_order

cart = sales.shopping_cart.Cart()
order = sales.shopping_order.Order()
order.get_input()

while not order.quit:
    cart.process(order)
    order = sales.shopping_order.Order()
    order.get_input()

print(cart)
```