ΠΕΡΙΕΧΟΜΕΝΑ:

Γιώργος Τασιούλης

Χρυσός Χορηγός Μαθήματος

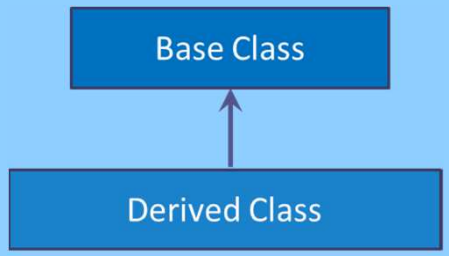
Πάνος Γ.
Αγγελική Γ. – Σταυριανή Γ.

Χρυσός Χορηγός Μαθήματος

1. Κληρονομικότητα

Η **κληρονομικότητα (inheritance):**

- Παίρνει μία κλάση, και την επαυξάνει σε μία καινούργια, προσθέτοντας σε αυτήν χαρακτηριστικά (μέλη – μεθόδους)
- Σχηματικά απεικονίζεται ως εξής:



  - Η **παραγόμενη κλάση (derived class)** περιέχει όλα τα μέλη και τις μεθόδους της **βασικής κλάσης (base class)**
  - και λέμε ότι η παραγόμενη κλάση (derived class) **κληρονομεί (inherits)** τη βασική κλάση (base class)
- Εναλλακτικά θα μπορούσε να αναπαρασταθεί και:

  - Για να φαίνεται ότι η παραγόμενη κλάση **επαυξάνει (επεκτείνει, extends)** τη βασική κλάση.

Ορίζουμε ότι η κλάση μας θα κληρονομήσει μία άλλη κλάση:

- θέτοντας το όνομα της βασικής κλάσης σε παρένθεση, που ακολουθεί το όνομα της παραγόμενης κλάσης:

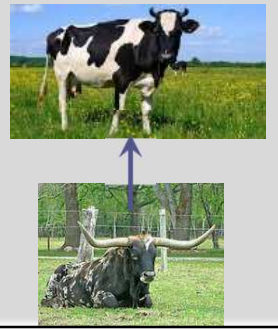
```
class Base:
    ...
class Derived(Base):
    ...
```

- ενώ ο κατασκευαστής της παραγόμενης κλάσης, μπορεί να καλέσει τον κατασκευαστή της βασικής κλάσης, ως εξής:

```
class Base:
    def __init__(self, b_attr):
        self.b_attr = b_attr
class Derived(Base):
    def __init__(self, b_attr, d_attr):
        super().__init__(b_attr)
        self.d_attr = d_attr
```

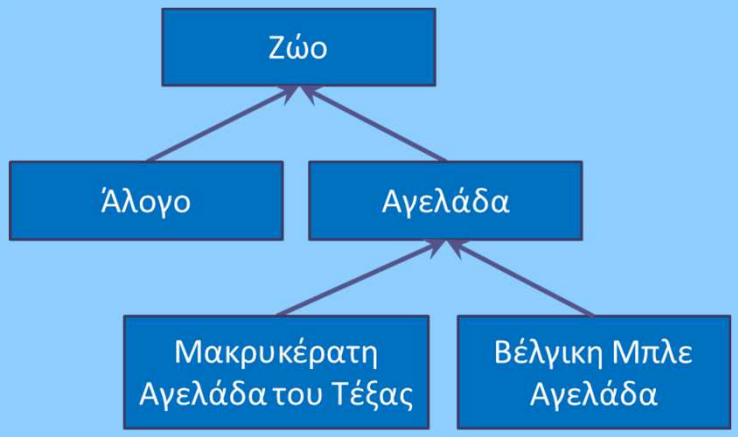
Παράδειγμα 1: inheritance.py

Ο Bob, η μακρυκέρατη αγελάδα του Τέξας, έχει κέρατα μήκους 0,5 μέτρου. Στο παράδειγμα επεκτείνουμε την κλάση αγελάδα, ώστε να μοντελοποιεί σωστά αυτόν τον τύπο αγελάδων και έπειτα τυπώνουμε τα χαρακτηριστικά του Bob.



<u>Η βασική σχέση που αναπαρίσταται με την κληρονομικότητα είναι η "είναι-ένα" (is-a)</u>:

- Με αυτήν μπορούν να οριστούν οντολογίες, όπως η εξής:



- και λέμε π.χ. ότι:
  - Το Ζώο είναι **υπερκλάση (superclass)** του Αλόγου
  - Η Αγελάδα είναι **υποκλάση (subclass)** του Ζώου

Για να φτιάξουμε οντολογίες σκεφτόμαστε:

- ποια είναι τα κοινά χαρακτηριστικά των αλόγων και των αγελάδων; Αυτά θα οριστούν στη βασική κλάση (Ζώο)
- Τι διαφοροποιεί την αγελάδα από το άλογο; Αυτά θα οριστούν στην παραγόμενη κλάση (Αγελάδα)

Σημείωση:

- Οι σχέσεις is-a και has-a είναι τα βασικά εργαλεία για την αναπαράσταση στον αντικειμενοστρεφή προγραμματισμό.

<u>Άσκηση 1:</u>

Ορίζουμε μια οντολογία του ζωϊκού βασιλείου:

- Ζώο: Έχει χαρακτηριστικά βάρος και ύψος
- Άλογο: Κληρονομεί το Ζώο και το επεκτείνει με:
  - το χρώμα του
  - την ούρά του (μήκος)
- Σκύλος: Κληρονομεί το Ζώο και το επεκτείνει με:
  - την ένταση γαβγίσματος (σε dB)
  - τη μέθοδο bark() που βγάζει έναν κατάλληλο ήχο
- Doberman: Κληρονομεί το Σκύλο και το επεκτείνει με:
  - τη μέθοδο run() που βγάζει κατάλληλο μήνυμα
- Bulldog: Κληρονομεί το Σκύλο και το επεκτείνει με:
  - το μέγεθος των αυτιών του
  - τη μέθοδο sleep() που βγάζει κατάλληλο μήνυμα

Έπειτα ορίστε ένα συγκεκριμένο άλογο, ένα Doberman και ένα bulldog:

- Τυπώστε το χρώμα του αλόγου
- Το Doberman γαβγίζει, τρέχει και μετά γαβγίζει
- Το Bulldog γαβγίζει, κοιμάται και μετα κοιμάται ξανα.

ΜΑΘΗΜΑ 18: OOP – Κληρονομικότητα 1.2. Υπέρβαση Μεθόδων (method overriding)

python 3 psounis

- Είναι συχνά χρήσιμο, η παραγόμενη κλάση **<u>να επαναόρίζει (override)</u>** κάποια μέθοδο της βασικής κλάσης.
- Το αντικείμενο της παραγόμενης κλάσης θα καλεί τη μέθοδο όπως αυτή έχει επαναοριστεί:

```
# method.overriding.py
class Cow:
   …
   def express(self):
      if self.hunger > 5:
         print("Mooooooowwwwwwwww")
      else:
         print("Mowww")

class TexasLonghorn(Cow):
   …
   def express(self):
      if self.hunger > 5:
         print("MEEoooEEEwwwwwwwww")
      else:
         print("MEoEwww")

molly = Cow(500, 10)
molly.express()

bob = TexasLonghorn(400,20,0.50)
bob.express()
…
```

<u>Άσκηση 2:</u>

Κατασκευάζουμε ένα πρόγραμμα που παρακολουθεί τα μουσικά μας βίντεο στο youtube:

- Ένα playlist στο youtube αποτελείται από το τίτλο του, την περιγραφή του, την χρονική διάρκειά του και τα βίντεο από τα οποία αποτελείται
- Ένα βίντεο χαρακτηρίζεται από το όνομα του καλλιτέχνη, το όνομα του κομματιού και την χρονική του διάρκεια.

Για το playlist να υπάρχουν οι μέθοδοι:

- Για την προσθήκη νέου βίντεο στο playlist
- __str__ που να επιστρέφει συμβολοσειρά με τα χαρακτηριστικά του playlist
- recommendation() να επιστρέφει τυχαία κάποιο βίντεο

Έπειτα η playlist να κληρονομείται από την classical_playlist:

- Προστίθεται το μέλος “περιόδος” στο οποίο αναπαρίσταται η χρονική περίοδος του playlist (π.χ. μπαρόκ)
- Επαναορίζει τη recommendation ώστε να επιστρέφει το 1ο βίντεο του playlist.

Στο κυρίως πρόγραμμα να κατασκευάζεται ένα playlist, ένα classical playlist, να τυπώνονται τα περιεχόμενά τους, και να γίνονται recommendations (ένα από το κάθε playlist)

- Αν κάνουμε override κάποια μέθοδο και θέλουμε για κάποιο λόγο, **στην παραγόμενη κλάση να κάνουμε κλήση στη μέθοδο της βασικής κλάσης που κάναμε override**
- Τότε χρησιμοποιούμε τη built-in συνάρτηση **super**(). Π.χ.:

```
# super.py
class Base:
    def __init__(self, b_attr):
        self.b_attr = b_attr
    def __str__(self):
        return "Some info about the base class"
class Derived(Base):
    def __init__(self, b_attr, d_attr):
        super().__init__(b_attr)
        self.d_attr = d_attr
    def __str__(self):
        return "Some info about the derived class"
    def giga_info(self):
        return super().__str__() + " AND " + self.__str__()
d = Derived(1,2)
print(d.giga_info())
```

- Το ίδιο αποτέλεσμα πετυχαίνουμε αντί της super(), γράφοντας:
  - Το όνομα της βασικής κλάσης: εδώ **Base.__str__(self)**
  - Το συντακτικό **super(Derived, self).__str__(self)** (σαν να λέμε: φέρε την υπερκλάση της Derived).
- καλύτερος θεωρείται ο τρόπος της super (δεν έχει την αναφορά του ονόματος της βασικής κλάσης, ούτε ορίσματα)

Άσκηση 3:

Παίζουμε με μία ιεραρχία κλάσεων:

- Το ζώο (Animal) έχει τη μέθοδο make_sound που τυπώνει μία κενή συμβολοσειρά
- Η γάτα (Cat), κληρονομεί το ζώο, και έχει τη μέθοδο make_sound που τυπώνει “Meow”
- Η γάτα Ιμαλαϊών (HimalayanCat), κληρονομεί τη γάτα, έχει τη μέθοδο make_sound που τυπώνει τον ήχο της απλής γάτας ακολουθούμενη από “Miouw Miouw”
- Ο σκύλος (Dog), κληρονομεί το ζώο έχει τη μέθοδο make_sound που τυπώνει “Woof Woof”
- Το Doberman που κληρονομεί το σκύλο και δεν ορίζει τη μέθοδο make_sound
- Το KingDoberman που κληρονομεί το Doberman και κάνει τον ήχο του απλού σκύλου ακολουθούμενο από “WOOAAAAAAAAF”

Ορίστε έπειτα ένα αντικείμενο από κάθε κλάση και εκτυπώστε τον ήχο που αυτό κάνει.

Υπενθύμιση (από μάθημα 16):

- Όλα τα μέλη και οι μέθοδοι είναι **public (δημόσια)**. Άρα είναι ορατά έξω από την κλάση (και από τις υποκλάσεις).
- Ορίζουμε με «__» ότι ένα μέλος ή μέθοδος είναι **private (ιδιωτικό)**. Άρα δεν είναι ορατά έξω από την κλάση (ούτε από τις υποκλάσεις)

- Ένα **προστατευμένο μέλος ή μέθοδος (protected) δηλώνεται με μονό underscore, πριν από το όνομα της μεταβλητής** και:
  - Είναι **ορατό από τις υποκλάσεις** της κλάσης
  - και **δεν είναι ορατό έξω από την κλάση** (ίδια συμπεριφορά εκτός από τις υποκλάσεις, με τις ιδιωτικές μεταβλητές)

```
# protected.py
class Base:
    def __init__(self):
        self._bpr_attr = 1 # protected member
class Derived(Base):
    def __init__(self):
        super().__init__()
        print(self._bpr_attr) # works fine
d = Derived()
print(d._bpr_attr) # error?
```

Και θυμίζουμε ότι δεν θα χρησιμοποιούμε αυτά τα χαρ/κά:

- Μιας και "είμαστε όλοι ενήλικοι"

Παρατηρήσεις:

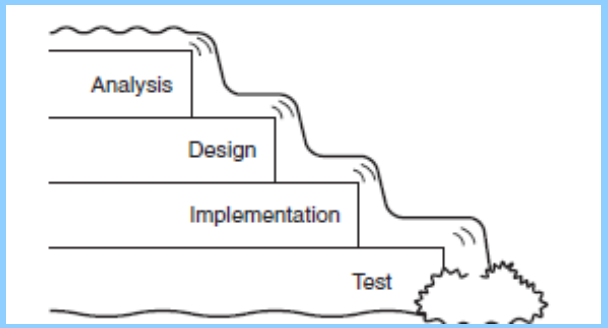
- Στην πραγματικότητα, είναι απλά μία σύμβαση.
- Αρκετοί προγραμματιστές βάζουν ένα underscore μπροστά από ένα μέλος της κλάσης, για να "πουν" σε άλλο προγραμματιστή που θα την επεκτείνει να μην την πειράξει!
- Αλλά ακόμη και οι "private" μεταβλητές είναι προσβάσιμες. π.χ. αν στην class Base έχουμε μία ιδιωτική μεταβλητή __var, τότε έχουμε πρόσβαση από οπουδήποτε στο πρόγραμμα ως obj._Base__var
- Γι' αυτό όσοι προέρχονται από τη C++ ή τη Java, να ξεχάσουν τα περί ενθυλάκωσης και να προσαρμοστούν στον κόσμο της Python!

Άσκηση 4:

Επαληθεύστε επεκτείνοντας το παράδειγμα της διαφάνειας ότι:

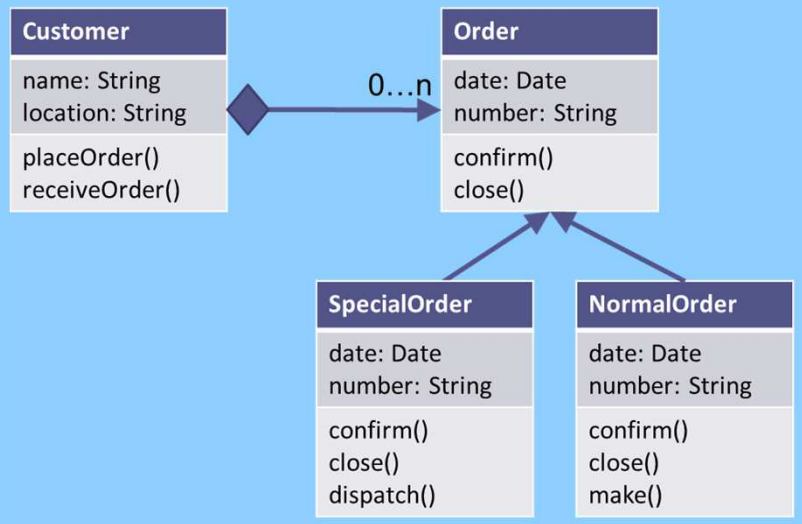
- Μπορούμε να έχουμε πρόσβαση και σε μία ιδιωτική μεταβλητή με τον τρόπο που αναφέρεται παραπάνω.

- Τα βασικά εργαλεία **μοντελοποίησης** του πραγματικού κόσμου είναι οι σχέσεις “είναι-ένα” (is-a) και “έχει-ένα” (has-a):
  - Είναι-ένα: Μοντελοποιείται με την κληρονομικότητα
  - Έχει-ένα: Μοντελοποιείται με αντικείμενο ως μέλος κλάσης.
- Για την κατασκευή μιας μεγάλης εφαρμογής (που απαιτεί συστηματική μοντελοποίηση) έχουν προταθεί διάφορα μοντέλα
  - Ένα δημοφιλές μοντέλο είναι “το μοντέλο του καταρράκτη”
  - στο οποίο **η ανάλυση και ο σχεδιασμός** είναι τα πρώτα που γίνονται προτού γραφεί έστω και μία γραμμή κώδικα.



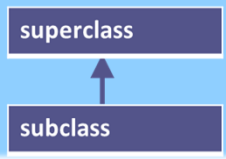
  - Η **Ανάλυση Συστημάτων** είναι η θεματική περιοχή η οποία ασχολείται με το πως μοντελοποιούμε ένα σύστημα του πραγματικού κόσμου.
  - Από τα αποτελέσματα της ανάλυσης είναι το **διάγραμμα κλάσεων**, στο οποίο φαίνονται οι σχέσεις μεταξύ των κλάσεων της εφαρμογής
  - Ενώ υπάρχουν ειδικές **γλώσσες προδιαγραφών (όπως η UML)** που ορίζουν π.χ. πως ζωγραφίζεται το διάγραμμα κλάσεων

Παράδειγμα Διαγράμματος Κλασεων:



- **Τα κουτάκια αναπαριστούν τις κλάσεις (όνομα-μέλη-μέθοδοι)**
- **Σχέση “έχει”: Αναπαρίσταται ως:**
  - με το βελάκι με το ρόμβο
  - και ο συμβολισμός μας λέει ότι ένας πελάτης “έχει” από 0…n παραγγελίες

- **Σχέση “είναι”: Αναπαρίσταται ως:**
  - με το απλό βελάκι.



- Η Ανάλυση Συστημάτων είναι ολόκληρο επιστημονικό πεδίο και ξεφεύγει από τα όρια αυτής της σειράς.
- Ωστόσο θα χρησιμοποιήσουμε σε κάποιες ασκήσεις, το διάγραμμα κλάσεων για λόγους διευκόλυνσης.

Άσκηση 5.1:

Η κλάση Customer έχει:

- Ονοματεπώνυμο
- Διεύθυνση
- Παραγγελίες (λίστα από αντικείμενα τύπου Order)
- Μέθοδος: place_order() τοποθετεί ένα αντικείμενο τύπου order στο τέλος της λίστας.
- __str__: Τυπώνει τα στοιχεία του, κάθε παραγγελία και το συνολικό ποσό των παραγγελιών του.

Η κλάση Order έχει μέλη:

- Ημερομηνία (συμβολοσειρά σε format “YYYYMMDD” π.χ. “20201105”)
- Πληρωμή: Αντικείμενο τύπου Payment

Η κλάση Payment έχει μέλη:

- Ποσό

Στο κυρίως πρόγραμμα:

- Ορίστε έναν πελάτη
- Κατασκευάστε τρεις παραγγελίες και θέστε τις στον πελάτη.
- Τυπώστε τον πελάτη (μέσω της __str__)

[Κατασκευάστε όποιαδήποτε επιπλέον μέθοδο κρίνετε σκόπιμη]

Άσκηση 5.2:

Η κλάση Payment κληρονομείται από την Credit η οποία έχει μέλη:

- number (αριθμός κάρτας)
- exp_date (ημ/νία λήξης κάρτας - συμβολοσειρά)

Η κλάση Payment κληρονομείται από την Check η οποία έχει μέλη:

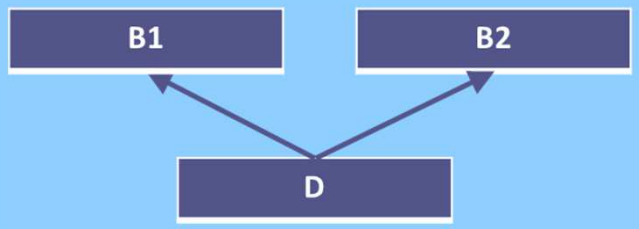
- number (αριθμός check_book)
- bank_code (συμβολοσειρά)

Στο κυρίως πρόγραμμα:

- Ορίστε έναν πελάτη
- Κατασκευάστε τρεις παραγγελίες (μία κανονική, μία με Credit και μία με Check) στον πελάτη
- Εκτυπώστε τον πελάτη (με την str)

[Ορίστε οτιδήποτε επιπλέον κρίνετε σκόπιμο]

- Μπορούμε να ορίσουμε ότι μία κλάση κληρονομεί από περισσότερες από μία κλάση, π.χ. από 2 κλάσεις:



- το συντάσσουμε ενθέτοντας στις παρενθέσεις τις κλάσεις από τις οποίες κληρονομεί:

```
# multiple.inheritance.py
class Base1:
    def __init__(self, b1_attr):
        self.b1_attr = b1_attr

class Base2:
    def __init__(self, b2_attr):
        self.b2_attr = b2_attr

class Derived(Base1, Base2):
    def __init__(self, b1_attr, b2_attr, d_attr):
        Base1.__init__(self, b1_attr)
        Base2.__init__(self, b2_attr)
        self.d_attr = d_attr

d = Derived(1,2,3)
print(f"{d.b1_attr}, {d.b2_attr}, {d.d_attr}")
```

- [Προσέξτε την χρήση Base1 και Base2 για την ενεργοποιήση των initializers].

Άσκηση 6:

Ο Βασιλιάς:

- έχει ένα βασίλειο (συμβολοσειρά)
- διοικεί (rule) τυπώνοντας “Now, I rule”

Ο Φιλόσοφος:

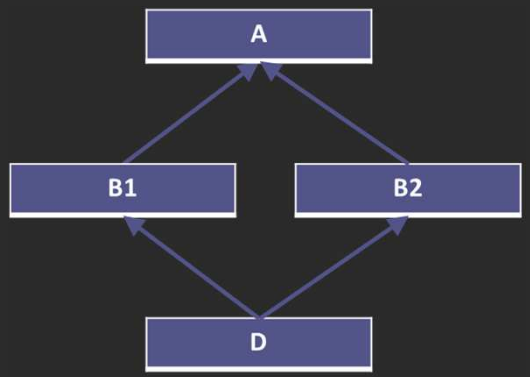
- ανήκει σε μία φιλοσοφική σχολή (συμβολοσειρά)
- σκέφτεται (think) τυπώνοντας “Now, I think”

Ο Μάρκος Αυρήλιος (121-180 μ.Χ.) ήταν αυτοκράτορας της Ρωμαϊκής Αυτοκρατορίας, που με το έργο του “Στοχασμοί” ανέδειξε χαρακτηριστικά της Στωϊκής φιλοσοφίας.
Η καθημερινή του ρουτίνα ήταν: σκεφτόταν, διοικούσε και μετά σκεφτόταν.

Προσομοιώστε μία μέρα της ζωής του Μάρκου Αυρήλιου με κατάλληλο πρόγραμμα Python.

- Σε περίπλοκα σχήματα πολλαπλής κληρονομικότητας δημιουργούνται αρκετά προβλήματα στην κατανόηση:
  - π.χ. τι γίνεται αν δύο υπερκλάσεις έχουν την ίδια μέθοδο ή το ίδιο μέλος; Ποια μέθοδος θα κληθεί από την τελική παραγόμενη κλάση;
  - Τι γίνεται αν οι δύο γονείς κληρονομούν από κοινή υπερκλάση (γνωστό ως το πρόβλημα του διαμαντιού)
- Η Python έχει υλοποιήσει ένα πολύ απλό μοντέλο:
  - **Πρώτα σειριοποιεί (βάζει σε μια σειρά – serialize) όλους τους προγόνους της κλάσης**, χρησιμοποιώντας έναν αλγόριθμο που καλείται **C3 – linearization**.
  - Από αυτόν αρχικοποιείται μια "σειρά των κλάσεων" που καλείται **"MRO – Method Resolution Order"** και έχουμε πρόσβαση σε αυτήν από τη μέθοδο mro() της κλάσης
  - Το τελικό αντικείμενο θα περιέχει ένα στιγμιότυπο από κάθε κλάση

```
# mro.py
class A:
   pass
class B1(A):
   pass
class B2(A):
   pass
class D(B1, B2):
   pass
print(D.mro())
```



- Με βάση αυτήν τη σειρά:
  - Αναζητείται από αριστερά προς τα δεξιά το χαρακτηριστικό και καλείται η πρώτη εμφάνισή του.
- Ειδικά μάλιστα για τη super()
  - Αυτή θα αντικατασταθεί από την κλάση που είναι ακριβώς στα δεξιά της με βάση την MRO.

Παράδειγμα 2: super.mro.py

```
class A:
   def __init__(self):
      print("Entering A")
      super().__init__()
      print("Exiting A")

class B1(A):
   def __init__(self):
      print("Entering B1")
      super().__init__()
      print("Exiting B1")
```

```
class B2(A):
   def __init__(self):
      print("Entering B2")
      super().__init__()
      print("Exiting B2")

class D(B1, B2):
   def __init__(self):
      print("Entering D")
      super().__init__()
      print("Exiting D")

D()
```

Παρατήρηση 1:

- Ο αλγόριθμος C3-linearization είναι αρκετά περίπλοκος, αλλά θέτει δύο προτεραιότητες:
  - Κάθε παιδί να προηγείται των γονέων του
  - Να σέβεται τη σειρά των γονέων όπως αυτές εμφανίζονται, όταν τα παιδιά τους κληρονομούν.
  - [google C3-linearization]

Παρατήρηση 2:

- Τα ίδια ισχύουν και για την απλή κληρονομικότητα.
- Και εκεί υπολογίζεται το MRO, αλλά εκεί προκύπτει η προφανής διάταξη παιδί->γονέας->παππούς

Παρατήρηση 3:

- Όλες οι κλάσεις κληρονομούν από την κλάση object.
- που περιέχει μεταξύ άλλων όλες τις dunder methods (χωρίς σώμα) ώστε να μπορούμε να τις επαναορίσουμε..

Παρατήρηση 4:

- Ο τρόπος που λειτουργέι η πολλαπλή κληρονομικότητα στην Python έχει επικριθεί έντονα
- Το μοντέλο με την MRO δεν ικανοποιεί πολλούς προγραμματιστές.
- ...Ωστόσο η πολλαπλή κληρονομικότητα πάντα έχει προβλήματα. Άλλες γλώσσες προγραμματισμού, σκόπιμα την παραλείπουν.

Άσκηση 7:

- Ορίστε την κλάση Person με μέλη το όνομα και το μισθό
- Ορίστε την κλάση Waiter:
  - να κληρονομεί την Person
  - να έχει μία μέθοδο serve (με ορίσματα πόσους πελάτες εξυπηρετεί και τον barista στον οποίο μεταφέρει την παραγγελία
- Ορίστε την κλάση Barista:
  - Να κληρονομεί την Person
  - Να έχει μία μέθοδο prepare (καλείται από την serve του Waiter)
- Ορίστε την κλάση Owner:
  - Κληρονομεί την Waiter και την Barista (κάνει και τις δύο δουλειές)
- Στο κυρίως πρόγραμμα:
  - Ορίζονται ένας ιδιοκτήτης, δύο σερβιτόροι και ένας μπαρίστα.
  - Ορίζονται δύο λίστες: Η πρώτη (waiters) να περιέχει όλους τους σερβιτόρους και η δεύτερη (baristas) να περιέχει όλους τους barista.
  - Έπειτα επαναληπτικά (10 φορές): Ερχεται μια παρέα (τυχαία από 1 έως 5 άτομα), επιλέγεται τυχαία ένας σερβιτόρος, ο οποίος επιλέγει τυχαία έναν από τους μπαρίστα.
  - Στο τέλος όλοι να τυπώνουν πόσους εξυπηρέτησε ο καθένας

- **Αφηρημένη κλάση (abstract class):**
  - **είναι μία κλάση, που δεν μπορούμε να κατασκευάσουμε αντικείμενά της.**
- Οι αφηρημένες κλάσεις είναι χρήσιμες:
  - Όταν δεν έχει νόημα να κατασκευάσουμε στιγμιότυπά της.
  - Και θέλουμε να ορίσουμε κάποιες μεθόδους τις οποίες οι παραγόμενες κλάσεις, θα πρέπει να επαναορίσουν.
- Μία **αφηρημένη μέθοδος (abstract method)** είναι μία μέθοδος που το σώμα της δεν ορίζεται.
- Για να ορίσουμε μία αφηρημένη κλάση:
  - Κάνουμε import το ABC (κλάση) από το module abc.
  - Η κλάση μας κληρονομεί αυτήν την κλάση
- Για να ορίσουμε μία αφηρημένη μέθοδο:
  - Βάζουμε το @abstractmethod πριν το όνομα της μεθόδου
  - Το οποίο είναι ένας decorator που επίσης πρέπει να κάνουμε import από το module abc.

```
# abstract.class.py
from abc import ABC, abstractmethod

class MyAbstractClass(ABC):
    def __init__(self, attr):
        self.attr = attr
    @abstractmethod
    def my_abstract_method(self):
        pass

ob = MyAbstractClass(1)
```

Άσκηση 8:
Τροποποιήστε την κλάση Animal της άσκησης 3, ώστε να είναι αφηρημένη κλάση.

Άσκηση 9:
Τροποποιήστε την κλάση Person της άσκησης 7, ώστε να είναι αφηρημένη κλάση.

Σημείωση:
- Οι decorators είναι προγραμματιστικό εργαλείο που εμπλουτίζουν τη λειτουργία των συναρτήσεων – μεθόδων και θα δούμε αναλυτικά σε προχωρημένα μαθήματα.

- Τα interfaces είναι μια ιδέα που προέρχεται (κυρίως) από τη Java και τη C#
  - Σε αυτές τις γλώσσες δεν υποστηρίζεται πολλαπλή κληρονομικότητα, αλλά μόνο απλή κληρονομικότητα.
  - Η "πατέντα" που έχουν εφεύρει για να προσομοιώσουν την πολλαπλή κληρονομικότητα, είναι τα interfaces.
- **Τα interfaces (στη Java) είναι πακέτα μεθόδων, χωρίς σώμα.**
  - Μπορούμε να "αναγκάσουμε" μία κλάση να "υλοποιήσει" (implement) ένα ή περισσότερα interfaces.
  - Έτσι μία κλάση μπορεί να κληρονομεί το πολύ μία κλάση και να υλοποιεί οσαδήποτε interfaces.

- **Οι διεπαφές (interfaces) στην Python προσομοιώνονται μέσω αφηρημένων κλάσεων και μεθόδων.**

```
# interface.py
from abc import ABC, abstractmethod

class MyInterface(ABC):
    @abstractmethod
    def interface_method1(self):
        pass

    @abstractmethod
    def interface_method2(self):
        pass


class MyRealClass(MyInterface):    ...
```

Άσκηση 10:

Ορίστε το interface GeometricObjectInterface:
- Μέθοδος: area(): Επιστρέφει το εμβαδόν
- Μέθοδος: perimeter(): Επιστρέφει την περίμετρο

Ορίστε την κλάση Circle:
- Μέλος: radius (ακτίνα)
- Υλοποιεί το interface GeometricObjectInterface

Ορίστε το interface Resizable:
- Μέθοδος: resize(): Τροποποιεί το αντικείμενο

Ορίστε την κλάση ResizableCircle:
- Κληρονομεί την Circle
- Υλοποιεί το interface Resizable

ΜΑΘΗΜΑ 18: OOP – Κληρονομικότητα

5. Mixins

python 3 psounis

- **Τα mixins** προέρχονται από γλώσσες που επιτρέπουν πολλαπλή κληρονομικότητα(π.χ. Ruby). Μέσω αυτών γίνεται προσπάθεια να προσομοιωθεί η πολλαπλή κληρονομικότητα, αποφεύγοντας όμως τα προβλήματα της.
  - **είναι κλάσεις που περιέχουν μεθόδους για να χρησιμοποιηθούν από άλλες κλάσεις**
  - **Χωρίς όμως να πρέπει να κληρονομηθούν από άλλες κλάσεις.** Λέμε ότι ένα mixin ενσωματώνεται στην κλάση (αντί να κληρονομείται)
- Συχνά είναι λειτουργικότητα που είναι κοινή σε πολλές κλάσεις (όπως π.χ. η αποθήκευση σε JSON αρχεία)
- Σημαντική διαφορά σε σχέση με τα interfaces: Τα mixins υλοποιούν της μεθόδους, ενώ τα interfaces δεν τις υλοποιούν

- **Τα mixins στην Python υλοποιούνται μέσω της πολλαπλής κληρονομικότητας**
- (οπότε είναι μία προσομοίωση που δεν αποφεύγει τυχόν προβλήματα της πολλαπλής κληρονομικότητας)
- Δες στο παράδειγμα mixin.methods.py μία τεχνική για να δημιουργούνται μέθοδοι που συνδυάζουν λειτουργικότητες
- Δες και στο παράδειγμα 3, πως ορίζοντας μία σταθερή λειτουργικότητα σε 4 τελεστές σύγκρισης, αρκεί να ορίζουμε στην κλάση μας τους υπόλοιπους 2 για να λειτουργούν όλοι οι τελεστές σύγκρισης (όπως είδαμε ότι γίνεται ήδη αυτόματα στις αντίστοιχες dunder methods)

Παράδειγμα 3: mixin.py

```
# mixin.py
class ComparableMixin(object):
    """This class has methods which use `<=` and `==`,
    but this class does NOT implement those methods."""
    def __ne__(self, other):
        return not (self == other)
    def __lt__(self, other):
        return self <= other and (self != other)
    def __gt__(self, other):
        return not self <= other
    def __ge__(self, other):
        return self == other or self > other

class Integer(ComparableMixin):
    def __init__(self, i):
        self.i = i
    def __le__(self, other):
        return self.i <= other.i
    def __eq__(self, other):
        return self.i == other.i

print(Integer(3)<=Integer(4))
print(Integer(3)<Integer(4))
```

- (Τα παραδείγματα της διαφάνειας είναι από: https://stackoverflow.com/questions/533631/what-is-a-mixin-and-why-are-they-useful)

ΜΑΘΗΜΑ 18: OOP – Κληρονομικότητα Data Structures: Linked List (Συνδεδεμένη Λίστα)

python 3 psounis

Άσκηση 11.1: Λίστα

Η λίστα είναι μια δομή δεδομένων με γραμμική διάταξη, στην οποία κάθε κόμβος είναι ένα ζεύγος από δεδομένα και μία αναφορά στον επόμενο κόμβο (None, αν δεν υπάρχει επόμενος κόμβος)

Ορίστε την Κλάση Node (σε αρχείο linked_list.py):

- Μέλη: data και next
- Μέθοδος: init (me default στο next το None)

Ορίστε την Κλάση LinkedList (στο ίδιο αρχείο):

- Μέλος: head (αρχικοποιείται σε None)
- Μέθοδος empty: T/F ανάλογα με το αν η λίστα είναι άδεια
- Μέθοδος insert_start(data): Εισάγει έναν νέο κόμβο με περιεχόμενο data στην αρχή της λίστας
- Μέθοδος insert_after(node, data): Εισάγει έναν νέο κόμβο με περιεχόμενο data μετά από τον κόμβο node
- Μέθοδος delete_start(): Διαγράφει τον 1ο κόμβο και επιστρέφει τα δεδομένα του
- Μέθοδος delete_after(node): Διαγράφει τον κόμβο μετά από τον node και επιστρέφει τα δεδομένα του
- Μέθοδος __str__(): Τυπώνει τα περιεχόμενα της λίστας

Υλοποιήστε τα παραπάνω και διαπιστώστε ότι όλα πάνε καλά.

Άσκηση 11.2: OrderedList

Επεκτείνετε την κλάση LinkedList, ορίζοντας την OrderedList ως εξής:

- Μέθοδος insert(data): Να δέχεται ένα δεδομένο και να το εισάγει στη λίστα ταξινομημένο (σε αύξουσα σειρά)
- Μέθοδος delete(data): Να διαγράφει το δεδομένο που δέχεται ως όρισμα

Διαπιστώστε ότι όλα πάνε καλά με κατάλληλα παραδείγματα στο κυρίως πρόγραμμα.

Άσκηση 12.1: Τροποποίηση στην κλάση χαρακτήρας

Τροποποιήστε την κλάση "Character" προάγοντας σε μέλη τιμές που χρησιμοποιήσαμε στο πρόγραμμά μας:

- max_delay: Να τίθεται ίσο με 5
- attack_range: tuple με ελάχιστη και μέγιστη τιμή (3 και 11 αντίστοιχα)
- Κάνετε και τις απαραίτητες τροποποιήσεις στις μεθόδους που χρησιμοποιούν τα παραπάνω

Άσκηση 12.2: Κλάση "Μάγος"

Ορίστε την κλάση "Mage" η οποία κληρονομεί τον χαρακτήρα και:

- Το attack_range του πρέπει να είναι μεταξύ 8 και 17
- Έχει μία μεταβλητή mana (ετοιμότητα για να κάνει spell). Αρχική τιμή 100 και ορίζεται επίσης max_mana = 100 και θα πρέπει στο τέλος του γύρου να αυξάνεται το mana κατά 1.
- Μέθοδος lightning_spell. Δεν δέχεται όρισμα και επιστρέφει έναν τυχαίο αριθμό μεταξύ 30 και 50. Κοστίζει 55 mana.
- Επαναορίζει τη μέθοδο attack έτσι ώστε: (α) να υπολογίζεται το delay με τον ίδιο τρόπο (β) να ελέγχει αν υπάρχει δυνατότητα για lightning spell. Αν ναι, να επιτίθεται με αυτό, αλλιώς να κάνει συμβατική επίθεση.

Άσκηση 12.3: Κλάση "Tank"

Ορίστε την κλάση "Tank" η οποία να κληρονομεί τον χαρακτήρα και:

- Το attack_range του πρέπει να είναι μεταξύ 20 και 30
- Η max_health του (και η αρχική του health) να είναι διπλάσια από την κανονική (πολλαπλασιαστής 2)

Άσκηση 12.4: main()

Επεκτείνετε την ομάδα των orcs με ένα tank και την ομάδα των elves με έναν mage και βάλτε τους να σφαχτούνε.