ΠΕΡΙΕΧΟΜΕΝΑ:

Γιώργος Τασιούλης
Σμαραγδένιος Χορηγός Μαθήματος

Παναγιώτα Γ.
Χρυσός Χορηγός Μαθήματος

ΜΑΘΗΜΑ 17: OOP – Ειδικές Μέθοδοι 1.1 Μετατροπή σε Συμβολοσειρα: __str__()

Οι **ειδικές μέθοδοι κλάσεων (special ή magic ή dunder(=double underscore) methods):**

- Είναι μέθοδοι που έχουν ήδη οριστεί σε κάθε κλάση και μπορούμε να τους προσθέσουμε λειτουργικότητα.
- **Μέσω αυτών μπορούμε να προσθέσουμε λειτουργίες στην κλάση μας** όπως:
  - Να μετατρέπουμε μία κλάση σε συμβολοσειρά
  - Να επιτρέπουμε να γίνεται πρόσθεση δύο αντικειμένων
  - Να συγκρίνουμε με το '>' δύο αντικείμενα
  - και πολλά ακόμη…
- Λέμε ότι με αυτές τις μεθόδους επιτυγχάνουμε την **υπερφόρτωση τελεστών (operator overloading)**
  - δηλαδή τελεστές (όπως π.χ. το +) θα λειτουργούν και με επιπλέον τρόπους (θα κάνουν και πρόσθεση των αντικειμένων μας και όχι μόνο πρόσθεση ακεραίων, πραγματικών, συνένωση λιστών κ.λπ.)
- Κοινό χαρακτηριστικό τους είναι **ότι ξεκινάνε και τελειώνουν με διπλό underscore.**
- Και έχουμε ήδη μελετήσει δύο τέτοιες "μαγικές" μεθόδους:
  - την __init__() που κατασκευάζει ένα νέο αντικείμενο και καλείται αυτόματα όταν κατασκευάζουμε ένα αντικείμενο.
  - την __del__() που διαγράφει ένα αντικείμενο και καλείται αυτόματα όταν γράφουμε π.χ. del obj

Ειδική Μέθοδος για την μετατροπή σε συμβολοσειρά:

- **__str()__**
  - Μετατρέπει το αντικείμενο σε συμβολοσειρά
  - Χρησιμοποιείται όταν γράφουμε **str(obj)**

Παράδειγμα 1: str.py

```
class Time:
    def __init__(self, hour, minute, second):
        self.hour = hour
        self.minute = minute
        self.second = second

    def __str__(self):
        return f"{str(self.hour).zfill(2)}:" \
               f"{str(self.minute).zfill(2)}:" \
               f"{str(self.second).zfill(2)}"


t = Time(11,2,3)
print(t)
print("The time is " + str(t))
```

Οι **σχεσιακοί τελεστές(comparison operators)** υπερφορτώνονται με τις εξής ειδικές μεθόδους:

Τελεστής	Μέθοδος
==	__eq__(self, other)
!=	__ne__(self, other)
<	__lt__(self, other)
<=	__le__(self, other)
>	__gt__(self, other)
>=	__ge__(self, other)

- Γράφοντας π.χ. **obj1 > obj2**, αυτό "μεταφράζεται" στην κλήση **obj1.__gt__(obj2)** και το obj2 διοχετεύεται ως παράμετρος other.
- Πρέπει να επιστρέφουν T/F.

Σημειώση 1: Αυτόματοι Ορισμοί

- Το > ορίζει αυτόματα και το < (αφου α < b ⇔ b > a) και αντιστρ.
- Το >= ορίζει αυτόματα και το <= (αφού α<=β ⇔ β>=α) και αντ.
- Το == ορίζει αυτόματα και το != (αφού α==β ⇔ όχι α!=β) κ αντ.

Σημειώση 2: Με άλλους τύπους δεδομένων:

- Χρησιμοποιώντας την isinstance μπορούμε να συγκρίνουμε με άλλους τύπους:
- Π.χ. time == 2 ⇔ time.__eq__(2) και έλεγχος της other με την isinstance (κ εντοπίζεται και η αντιμεταθετικότητα)

Παράδειγμα 2: comparison.py

```
class Time:
   ### -- init, str ίδια με παράδειγμα 1 --
  def __gt__(self, other):
      if self.hour > other.hour:
         return True
      elif self.hour == other.hour:
         if self.minute > other.minute:
            return True
         elif self.minute == other.minute:
            if self.second > other.second:
               return True
      return False

t = Time(11,2,3)
t2 = Time(11,11,1)
print(f"{t} > {t2}: {t>t2}")
```

βλ. και: **comparison2.py** για τους αυτόματους ορισμούς τελεστών (σημείωση 1)

βλ. και: **comparison3.py** για σύγκριση με άλλους τύπους δεδομένων (σημείωση 2)

Άσκηση 1: Σύγκριση Γραμμών με Βάση το Μήκος τους

Επεκτείνουμε την κλάση "Γραμμή" (μάθημα 16, άσκηση 7):

- Ορίστε την ειδική μέθοδο __str__()
- Ορίστε τους απαραίτητους σχεσιακούς τελεστές, ώστε να μπορούμε να συγκρίνουμε με οποιονδήποτε σχεσιακό τελεστή δύο γραμμές (με βάση το μήκος τους)
- Μεριμνήστε ώστε όλες οι συγκρίσεις να δουλεύουν και όταν γίνεται σύγκριση με έναν ακέραιο αριθμό.

Άσκηση 2: Ταξινόμηση φοιτητών με βάση το βαθμό

Επεκτείνουμε το πρόγραμμα της άσκησης 5, μάθημα 16:

- Επεκτείνετε την κλάση φοιτητής, προσθέτοντας υπερφόρτωση του < (με βάση το βαθμό) και της str.

Ταξινομήστε τον πίνακα φοιτητών σε φθίνουσα σειρά (με τη sort) και τυπώστε τον.

Οι **διθέσιοι τελεστές (binary operators)** υπερφορτώνονται με τις εξής ειδικές μεθόδους:

Τελεστής	Μέθοδος
+	__add__(self, other)
-	__sub__(self, other)
*	__mul__(self, other)
//	__floordiv__(self, other)
/	__div__(self, other)
%	__mod__(self, other)
**	__pow__(self, other)
<<	__lshift__(self, other)
>>	__rshift__(self, other)
&	__and__(self, other)
|	__or__(self, other)
^	__xor__(self, other)

- Γράφοντας π.χ. **obj1 + obj2**, αυτό "μεταφράζεται" στην κλήση **obj1.__add__(obj2)** και το obj2 διοχετεύεται ως παράμετρος other.

Σημειώση 1: Επιστρεφόμενη Τιμή

- Πρέπει να επιστρέφουν ένα νέο αντικείμενο της κλάσης για την οποία ορίζεται

Σημειώση 2: Με άλλους τύπους δεδομένων

- Όπως με τους σχεσιακούς, π.χ. με την obj + other μεταφράζεται στην __add__(obj, other) οπότε με την is_instance διαχειριζόμαστε και άλλους τύπους δεδομένων

Σημειώση 3: Το αντίστροφο δεν ισχύει:

- Δηλαδή other + obj (όπου other αντικείμενο άλλου τύπου δεδομένων).
- Ωστόσο έχουν οριστεί ειδικές μέθοδοι για όλους τους δυαδικούς τελεστές που δουλεύουν όταν το obj είναι δεξί μέλος σε έναν τελεστή.
- Αυτές έχουν το ίδιο όνομα με ένα r μπροστά (π.χ. __radd__())

Παράδειγμα 3: binary.py

Σε αυτό επεκτείνεται η κλάση Time ώστε να προσθέτει δύο χρονικές στιγμές με την __add__ και τη __radd__

Άσκηση 3:

Ορίστε την κλάση Byte να περιέχει έναν πίνακα 8 ακεραίων (0 ή 1) (να αρχικοποιείται στο 0) και να ορίζει όλους τους δυαδικούς τελεστές επεξεργασίας bit καθώς και την __str__
Αναδείξτε τη λειτουργία με κατάλληλα παραδείγματα.

[Αυξημένης δυσκολίας άσκηση => βλ. βίντεο]

Οι **τελεστές καταχώρησης (augmented assignment operators)** ορίζονται με τις παρακάτω "μαγικές" μεθόδους:

Τελεστής	Μέθοδος
+=	__iadd__(self, other)
-=	__isub__(self, other)
*=	__imul__(self, other)
//=	__ifloordiv__(self, other)
/=	__idiv__(self, other)
%=	__imod__(self, other)
**=	__ipow__(self, other)
<<=	__ilshift__(self, other)
>>=	__irshift__(self, other)
&=	__iand__(self, other)
|=	__ior__(self, other)
^=	__ixor__(self, other)

- Γράφοντας π.χ. **obj1 += obj2**, αυτό "μεταφράζεται" στην κλήση **obj1.__iadd__(obj2)** και το obj2 διοχετεύεται ως παράμετρος other.

Σημειώση 1: Επιστρεφόμενη Τιμή

- Πρέπει να τροποποιεί το αντικείμενο self και να το επιστρέφει.

Σημειώση 2: Με άλλους τύπους δεδομένων

- Όπως με τους σχεσιακούς, π.χ. με την obj + other μεταφράζεται στην __iadd__(obj, other) οπότε με την is_instance διαχειριζόμαστε και άλλους τύπους δεδομένων

Άσκηση 4:

Ορίστε την κλάση Point3D με τρεις συντεταγμένες x,y,z

- Ορίστε την __str__()
- Ορίστε την __add__()
- Ελέγξτε την ορθότητα του κώδικα σας κάνοντας τις πράξεις σε αντικείμενα, με τον εξής κώδικα (exercise04.initial.py):

```
a = Point3D(1,1,1)
b = Point3D(2,2,2)
print(a+b)
c = a + b
print(c)
a = a + b
print(a)
a += b
print(a)
```

- Έπειτα ορίστε την __iadd()__ και παρατηρήστε τις διαφορές (αν υπάρχουν)

Οι **μονοθέσιοι τελεστές (unary operators)** χωρίζονται σε δύο κατηγορίες. Τους **τελεστές που χειρίζονται πράξεις**:

Τελεστής	Μέθοδος
-	__neg__(self)
+	__pos__(self)
~	__invert__(self)
abs()	__abs__(self)
round()	__round__(self[,ndigits])
floor()	__floor__(self)
ceil()	__ceil__(self)

(Οι round και abs είναι built-in στην Python. Οι floor και ceil είναι συναρτήσεις που έχουν οριστεί στο module math.)

και τους **τελεστές που κάνουν μετατροπή σε στοιχειώδεις τύπους δεδομένων:**

Τελεστής	Μέθοδος
int()	__int__(self)
float()	__float__(self)

(εδώ μπορεί να υπάχθεί και η μετατροπή σε string με την __str__())

Άσκηση 5:

Επεκτείνετε την κλάση Byte της άσκησης 3 με την invert (~)

Άσκηση 6:

Ορίστε μία κλάση με όνομα Counter η οποία θα υλοποιεί έναν απλό μετρητή (μέλος: μια ακέραια μεταβλητη) και θα έχει τις μαγικές μεθόδους:

- __str__: Επιστρέφει την τιμή του μετρητή
- __pos__: Αυξάνει την τιμή του μετρητή κατά 1.
- __neg__: Μειώνει την τιμή του μετρητή.

Άσκηση 7:

Ορίστε μία κλάση με όνομα Length η οποία θα απεικονίζει κάποιο μήκος (σε διαφορετικά συστήματα μέτρησης).

- __init__: Παίρνει δύο ορίσματα, ένα αριθμητικό και μία συμβολοσειρά που μπορεί να είναι “m”, “cm” ή “in”. Να αποθηκεύει τη μετατροπή σε μέτρα της αντίστοιχης ποσότητας (1 inch = 0,0254m)
- __str__: Τύπώνει σε μορφή π.χ. 0.34m
- __add__ : Με την προφανή χρήση.
- __round__: Στρογγυλοποιεί με χρήση της math.round(ndigits) όπου ndigits το δεκαδικό ψηφίο στο οποίο θα γίνει η στρογγυλοποίηση.
- __int__: Μετατρέπει τον αριθμό σε ακέραιο (αποκόπτοντας το δεκαδικό μέρος)

Ορίζοντας τις παρακάτω δύο ειδικές μεθόδους:

Τελεστής	Μέθοδος
len()	__len__(self)
[pos]	__getitem__(self, pos)

Μπορούμε να ορίσουμε **επανάληψη επί ενός αντικειμένου** (το οποίο προφανώς πρέπει να περιέχει κάποια δομή που να έχει νόημα η επανάληψη)

Παράδειγμα 3: iterate.py

```
class X:
    def __init__(self):
        self.ar = [x for x in range(5)]
    def __len__(self):
        return len(self.ar)
    def __getitem__(self, item):
        return self.ar[item]

obj = X()
for item in obj:
    print(item)
```

Ενώ με την ειδική μέθοδο **setitem** μπορούμε να γράφουμε π.χ a[i]=1

Τελεστής	Μέθοδος
[pos]	__setitem__(self, pos, value)

Άσκηση 8:

Επεκτείνετε την κλάση Byte της άσκησης 5, ώστε να υποστηρίζει επανάληψη και ανάθεση επί ενός συγκεκριμένου bit του Byte

Άσκηση 9:

Επεκτείνετε την κλάση Point3D της άσκησης 4, έτσι ώστε για ένα αντικείμενο obj της κλάσης:

- obj[0]: να επιστρέφει το x
- obj[1]: να επιστρέφει το y
- obj[2]: να επιστρέφει το z

Να είναι εφικτή και η ανάθεση τιμής στην αντίστοιχη συντεταγμένη, με την παραπάνω πρόσβαση.

Ορίζοντας την παρακάτω ειδική μέθοδο:

Τελεστής	Μέθοδος
(params)	__call__(self, parameters)

Το αντικείμενο μετατρέπεται σε συνάρτηση(!) και μπορεί να κληθεί π.χ. ως obj(parameters)

Παράδειγμα 4: function.py

```
class Polynomial:
    def __init__(self, *coeff):
        self.coeff = [c for c in coeff]
        print(self.coeff)
    def __str__(self):
        st = []
        for i in range(len(self.coeff)):
            st.append(f"{self.coeff[i]}*x^{len(self.coeff)-i-1}")
        return " + ".join(st)
    def __call__(self, x):
        res = 0
        for i in range(len(self.coeff)):
            res += self.coeff[i] * x ** (len(self.coeff)-i-1)
        return res
p = Polynomial(5,1,2)
print(str(p))
print(p(2))
```

Άσκηση 10:

Ένας κύκλος με κέντρο την αρχή των αξόνων περιγράφεται από την εξίσωση: $x^2 + y^2 = c^2$. Ορίστε την κλάση Cycle στην οποία να ορίζονται:

- __init__: να αρχικοποιεί το c
- __str__: να τυπώνει την εξίσωση του κύκλου
- __eq__: να εξετάζει αν δύο κύκλοι είναι ίσοι
- __lt__: Να εξετάζει αν είναι μικρότερος (σε ακτίνα) από τον κύκλο στα δεξιά της ανίσωσης.
- __call__: Αν διοχετευτεί ένα όρισμα που είναι ακέραιος ή float, αυτός να ερμηνεύεται ως x-συντεταγμένη και να υπολογίζονται τα σημεία του κύκλου που βρίσκονται σε αυτή τη συντεταγμένη και να επιστρέφονται. Αν διοχετευτούν δύο ορίσματα, τότε αυτό να ερμηνεύεται ως το ζεύγος συντεταγμένων του σημείου στο επίπεδο και να τυπώνεται αν ανήκει “μέσα”, “πάνω” ή “έξω” από τον κύκλο.

Παρατήρηση:

- Στην πραγματικότητα όλες οι συναρτήσεις στην Python μεταφράζονται σε αντικείμενα.
- Έτσι όταν ορίζουμε μία συνάρτηση, αυτή μεταφράζεται σε αντικείμενο (που θα έχει μία call method) χωρίς άλλα μέλη.
- Το γεγονός ότι όλες οι συναρτήσεις είναι αντικείμενα έχει πολλές ενδιαφέρουσες χρήσεις που θα δούμε σε προχωρημένα μαθήματα.

Είδαμε ήδη ότι η :

- **__str__:** Επιστρέφει μια συμβολοσειρά που αναπαριστά το αντικείμενο.
- Όπως είναι αναμενόμενο, η __str__ πρέπει να επιστρέφει ένα string το οποίο θα απεικονίζει μια συμβολοσειρά **που απευθύνεται στον τελικό χρήστη της εφαρμογής.**

Ορίζεται επίσης η:

- **__repr__:** Επιστρέφει μια συμβολοσειρά που αναπαριστά το αντικείμενο.
- Ωστόσο αυτή η συμβολοσειρά απευθύνεται κυρίως στον προγραμματιστή (από εμάς, για εμάς) και απεικονίζει το αντικείμενο σε μορφή όνομα-αντικειμένου(τιμές ορισμάτων)

Παρατήρηση:

- Η repr (representation) είναι χρήσιμη στον προγραμματιστή κατά τη φάση του debugging.
- Αν δεν έχει υλοποιηθεί η __str__ τότε αυτόματα καλείται η __repr__ στη θέση της.
- Θεωρείται καλή προγραμματιστική πολιτική να υπάρχει πάντα η __repr__ σε κάθε κλάση που κατασκευάζουμε.

Παράδειγμα 5: repr.py

Οι μέθοδοι repr και str της time θα είναι οι εξής:

```
class Time:
    …
    def __str__(self):
        return f"{str(self.hour).zfill(2)}:" \
               f"{str(self.minute).zfill(2)}:" \
               f"{str(self.second).zfill(2)}"
    def __repr__(self):
        return f"Time({self.hour},{self.minute},{self.second})“
t = Time(11,1,2)
print(t)
print(repr(t))
```

Άσκηση 11

Ορίστε την κλάση Date με μέλη (day, month, year). Να υπάρχουν οι μέθοδοι:

- __init__
- __str__
- __repr__
- __eq__

Έπειτα ορίστε την κλάση DateTime η οποία να “έχει” ένα Date και ένα Time αντικείμενο. Να οριστούν η __str__ και η __repr__ της νέας κλάσης.

ΜΑΘΗΜΑ 17: OOP – Ειδικές Μέθοδοι 1.9 Δημιουργία και Καταστροφή Αντικειμένου

python 3 psounis

[Προχωρημένα Θέματα. Συμβουλευθείτε το Βίντεο]

Η δημιουργία ενός αντικειμένου γίνεται στην πραγματικότητα σε δύο στάδια:

- **__new__:** Δημιουργεί το αντικείμενο. Είναι μέθοδος της κλάσης
- **__init__:** Αρχικοποιεί το αντικείμενο. Είναι μέθοδος του αντικειμένου.

Συνεπώς είναι λάθος να αναφερόμαστε στην init ως «κατασκευαστή»

Παράδειγμα 6: new.py

```
class A:
    def __new__(cls):
        print("A.__new__ called")
        return super(A, cls).__new__(cls)
    def __init__(self):
        print("A.__init__ called")
A()
```

cls: Προσδιορίζει ότι η new είναι μέθοδος της κλάσης
super(A,cls).__new__(cls): Δημιουργεί ένα κενό αντικείμενο, το οποίο κατασκευάζεται ως στιγμιότυπο της κλάσης object (της οποίας τα χαρακτηριστικά κληρονομούν όλα τα αντικείμενα)

Παρατήρηση:

- Θα μάθουμε τι ακριβώς είναι η super() στο επόμενο μάθημα (κληρονομικότητα)

Διαχείριση μνήμης:

- Σε παλιότερες γλώσσες (C, C++) ήταν ευθύνη του προγραμματιστή **να διαχειρίζεται τη μνήμη** των αντικειμένων (δέσμευση – αποδέσμευση). Αυτό οδηγούσε συχνά σε προγραμματιστικά λάθη.
- Νεότερες γλώσσες (Java, Python) αυτοματοποιούν τη διαδικασία. Π.χ. με τη __new__ για τη δέσμευση και τον αυτόματο **συλλέκτη σκουπιδιών (garbage collector)** για την αποδέσμευση μνήμης.
- Ένα αντικείμενο είναι "σκουπίδι" αν δεν υπάρχει καμία αναφορά που να δείχνει σε αυτό. Η Python τότε αποδεσμεύει αυτόματα τον χώρο του (δεν το ελέγχουμε εμείς)
- Η **del** σβήνει το όνομα που σχετίζεται με κάποιο αντικείμενο.
- Προσοχή όμως ότι η μέθοδος **__del__ καλείται από τον garbage collector όταν το πλήθος των αναφορών που δείχνουν σε αυτό είναι ίσο με το 0.**

Συνεπώς είναι λάθος να αναφερόμαστε στη del ως τον "καταστροφέα" που έχουμε σε άλλες γλώσσες προγραμματισμού.

Παράδειγμα 7: del.py

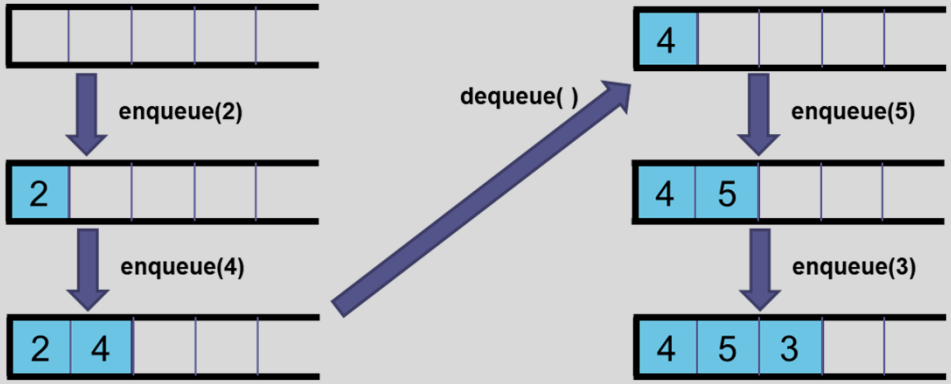
Στο παράδειγμα αυτό αναδεικνύεται η λειτουργία της del και ότι δεν καλείται αυτόματα.

Σημείωση:

- Εδώ κάπου σταματάμε, αλλά υπάρχουν και άλλες μέθοδοι και χαρακτηριστικά, τα οποία θα δούμε σε προχωρημένα μαθήματα.

Άσκηση 12.1: Ουρά

Η ουρά είναι μια δομή δεδομένων με γραμμική διάταξη, στην οποία η προσθήκη (enqueue) γίνεται στο τέλος της ούρας και η απομάκρυνση (dequeue) γίνονται από την αρχή της ουράς (FIFO: First In, First Out)



Ορίστε την Κλάση Queue (σε αρχείο queue.py):

- Περιέχει μία λίστα (με όνομα array)
- Αρχικοποιείται στην κενή λίστα
- Μέθοδος enqueue: Παίρνει όρισμα κάποιο στοιχείο και το βάζει στο τέλος του array
- Μέθοδος dequeue: Εξάγει το στοιχείο στην αρχή του array και το επιστρέφει. Επιστρέφει None αν η λίστα είναι άδεια.

Άσκηση 12.2: Dunder Methods

Επεκτείνετε την κλάση, με χρήση μαγικών μεθόδων, ώστε να επιτυγχάνεται η λειτουργικότητα:

- Μετατροπή σε string
- QueueObj + Obj (Obj: στοιχείο της ουράς)
- QueueObj += Obj
- -QueueObj
- len(QueueObj)

Άσκηση 12.3: Στα ταμεία της τράπεζας..

Μία τράπεζα αποτελείται από:

- Ν ταμεία (πίνακας από ουρές)

και τις ενέργειες:

- customer_enters(full_name): Αναθέτει τον πελάτη σε ένα από τα πέντε ταμεία στην τύχη
- customer_served(): Ελέγχει ποια ταμεία έχουν πελάτη και εξυπηρετεί κάποιον στην τύχη. Ο πελάτης αφαιρείται από την αντίστοιχη ουρά.
- Μετατροπή σε String: Εκτυπώνει τα ταμεία και τους πελάτες που είναι στην ουρά.

Στο κυρίως πρόγραμμα η τράπεζα έχει 3 ταμεία:
Συνολικά 100 φορές:

- Καταφθάνει πελάτης (70% πιθανότητα)
- Εξυπηρετείται πελάτης (30% πιθανότητα)

Ανά 10 φορές να εκτυπωθεί και η τρέχουσα κατάσταση στα ταμεία.

Άσκηση 13.1: Dunder methods

Ορίστε στην κλάση Character:

- Το νεό χαρακτηριστικό max_health που να αρχικοποιείται σε 100.

Και τις dunder methods:

- __str__()
- __repr__()
- και αυτές που απαιτούνται για να μπορούμε να γράφουμε obj +=h ή obj -= h οι οποίες να προσθέτουν ή να αφαιρούν αντίστοιχα από την υγεία του χαρακτήρα την ποσότητα h

και στην κλάση Arena τις dunder methods:

- __str__()
- __repr__()

Άσκηση 13.2: Κλάση Equipment

Περιέχει:

- sword: Πραγματικός, που είναι από 1.1-1.5. Είναι πολλαπλασιαστής με τον οποίο θα πολλαπλασιάζεται η επίθεση του χαρακτήρα
- cape: Πραγματικός, που είναι από 1.1-1.3. Είναι πολλαπλασιαστής με τον οποίο θα πολλαπλασιάζεται ο δείκτης υγείας του χαρακτήρα.

Κάθε χαρακτήρας θα περιέχει ένα αντικείμενο τύπου Equipment και γίνεται κατάλληλη αρχικοποίηση στους δείκτες υγείας και ταχύτητας επίθεσης.

Κάντε τις απαραίτητες τροποποιήσεις στον χαρακτήρα, έτσι ώστε:

- Η υγεία του χαρακτήρα να απεικονίζεται εσωτερικά με πραγματικό αριθμό, αλλά όλες οι εκτυπώσεις να γίνονται με στρογγυλοποίηση στον πλησιέστερο ακέραιο.
- Η επίθεση ενός παίκτη να είναι πάντα ακέραιος αριθμός (ο πλησιέστερος)

Εκτελέστε το πρόγραμμα με την ίδια σύνθεση ομάδων με την προηγούμενη έκδοση, αλλά πλέον όλοι οι χαρακτήρες θα έχουν και εξοπλισμό που θα αρχικοποιείται με τυχαίο τρόπο

[Σημείωση: Στο module random, υπάρχει η συνάρτηση uniform(start,stop) που παράγει τυχαίους πραγματικούς στο έυρος [start, stop]

Άσκηση 14.1: Κλάση Pupil (pupil.py)

Μέλη:

- Ίδια με αυτά του λεξικού. Κάντε μετονομασία σε first_name, last_name και pupil_id στα κλειδιά (και στο JSON αρχείο)

Μέθοδοι:

- __init__ : Να αρχικοποιεί μέσω ορισμάτων ένα αντικείμενο. Να έχει και default τιμές ορισμάτων.
- from_dict: Παίρνει όρισμα ένα λεξικό (όπως στο JSON) και αρχικοποιεί το αντικείμενο
- to_dict: Επιστρέφει το λεξικό που απεικονίζει το αντικείμενο.
- __str__(): Εκτυπώνει μορφοποιημένα έναν μαθητή.

[Σημείωση: Αλλάξτε και την print_teacher στην αντίστοιχη dunder method]

Άσκηση 14.2: Κλάση Pupils (pupil.py)

Μέλη:

- Περιέχει λίστα με όνομα pupils (αντικείμενα τύπου Pupils)

Μέθοδοι:

- __init__: Διαβάζει από το αρχείο JSON και αποθηκεύει στη λίστα τους μαθητές
- __str__: Εκτυπώνει μορφοποιημένα τους μαθητές.
- save_pupils_data(): Σώζει στο αρχείο JSON τους μαθητές (ως λεξικά)
- next_id(): Βρίσκει το επόμενο id

Και με τις ίδιες προδιαγραφές όπως στην προηγούμενη έκδοση:

- create_pupil(): Με τις ίδιες προδιαγραφές όπως πριν
- search_pupil_by_id(pupil_id)
- search_pupil_by_surname(last_name)
- pupil_update(pupil_id)
- delete_pupil_by_id(pupil_id)
- print_pupils_names()

Κάντε τις απαραίτητες διορθώσεις και στη main()

"There should be one – and preferably only one -- obvious way to do it."

Zen of Python #13