



ΠΕΡΙΕΧΟΜΕΝΑ:

1. Συναρτήσεις
 1. Παράμετροι
 2. Επιστρεφόμενες Τιμές
 3. Καθολικές και Τοπικές Μεταβλητές
 4. Οργάνωση Προγράμματος
 5. Κλήση μέσω αναφοράς αντικειμένου
2. Debugging Part 3: PyCharm
3. Game Project: Black Jack
4. Data Project: CRUD - Read

Κώστας Μ.

Χρυσός Χορηγός Μαθήματος

Νεκτάριος Χρήστου

Ασημένιος Χορηγός Μαθήματος

- Μία **Συνάρτηση (Function)** είναι:

- Ένα τμήμα κώδικα με ένα όνομα το οποίο έχει σχεδιαστεί για να κάνει μια συγκεκριμένη δουλειά

Ορισμός συνάρτησης:

- Ακολουθούμε το πρότυπο:

```
def func_name(param1, param2,...):
    ...
    ...
    return value(s)
```

- def:** ορίζει ότι θα ακολουθήσει ορισμός συνάρτησης
- func_name:** **όνομα** το οποίο ορίζουμε εμείς
- parameters:** Είναι οι παράμετροι της συνάρτησης [0..n]
- Ακολουθεί το **σώμα** με τις εντολές της συνάρτησης
- Στο σημείο που επιθυμούμε (συνήθως στο τέλος) ορίζουμε τις επιστρεφόμενες τιμές [0...n] με τη **return** (προσοχή στην : και στο ένα επίπεδο indentation)
- Ενώ η **κλήση** της συνάρτησης, γίνεται (μετά τον ορισμό της) ως

```
func_name(arg1, arg2,...)
```

 - διοχετεύοντας τιμές στα ορίσματα (**arguments**)

Παράδειγμα 1:

function.definition.py

Μία συνάρτηση χωρίς παράμετρους και επιστρεφόμενες τιμές.

```
def hello():
    print("Hello World!")
hello()
hello()
```

Η ιδέα προέρχεται από τις μαθηματικές συναρτήσεις

- Ορολογία στα μαθηματικά: π.χ. στη συνάρτηση $f(x) = 5x + 1$,
 - το f είναι το **όνομα** της συνάρτησης,
 - το x είναι το **όρισμα** της συνάρτησης και
 - το $5x + 1$ είναι το **σώμα** της συνάρτησης.
- Χρήση της συνάρτησης: Π.χ. με όρισμα το 15, το $f(15)$ (λέμε ότι **καλούμε** την f με όρισμα το 15)
 - υπολογίζεται** $5*15+1=76$ και λέμε ότι **επιστρέφεται** το 76.

Γιατί και πότε χρησιμοποιούμε συναρτήσεις:

- Για να μη γράφουμε τον ίδιο κώδικα πολλές φορές. Όταν βλέπουμε ότι έχουμε γράψει ίδιο ή παρόμοιο κώδικα πολλές φορές, τότε η χρήση συνάρτησης είναι μάλλον επιβεβλημένη.
- Για να σπάμε το πρόγραμμα σε μικρότερα κομμάτια, οπότε θα είναι ευκολότερο να γράψουμε, να διαβάσουμε και να ελέγξουμε τον κώδικά μας (Αποφεύγουμε δηλαδή, τον λεγόμενο κώδικα-μακαρόνι)
- Για να δομήσουμε επαναλαμβανόμενες ενέργειες που είναι εννοιολογικά αυτόνομες (π.χ. εισαγωγή εγγραφής, διαγραφή εγγραφής, εκτύπωση δεδομένων)

Παράμετροι

Μπορούμε να ορίσουμε όσες παράμετρους θέλουμε σε μία συν/ση.

- Η κάθε παράμετρος μπορεί να είναι οποιουδήποτε τύπου δεδομένων.
- Προσοχή όμως, ότι στην κλήση της συνάρτησης θα πρέπει να έχουμε διοχετεύσει όρισμα σε κάθε μία από τις παραμέτρους.
- και η συσχέτιση είναι η προφανής: Το όρισμα 1 αντιστοιχεί στην παράμετρο 1, το όρισμα 2 στην παράμετρο 2 κ.ο.κ.

```
def func_name(param1, param2,...):
    ...
    ...
    func_name(arg1, arg2,...)
```

Παράδειγμα 2: parameters.py

```
def print_full_name(name, surname, fathers_name):
    print(f"{name} {fathers_name[0]}. {surname}")

print_full_name("Bruce", "Wayne", "Thomas")
print_full_name("Clark", "Kent", "Jonathan")
```

Παρατήρηση:

Το PEP8 (στυλ) ορίζει ότι πρέπει να προηγούνται και να έπονται δύο κενά από τον ορισμό της συνάρτησης.

Παρατήρηση:

Η κλήση της συνάρτησης πρέπει πάντα να έπεται του ορισμού της συνάρτησης: Π.χ. ο κώδικας στα αριστερά δεν δουλεύει ενώ ο κώδικας στα δεξιά δουλεύει:

```
f()

def f():
    pass
```

```
def f():
    pass

f()
```

και αν και μπορούμε να ορίσουμε οπουδήποτε στο πρόγραμμά μας μια συνάρτηση, είναι καλό πρώτα να γράφουμε τις συναρτήσεις και μετά να ακολουθούν οι εντολές του προγράμματος.

Άσκηση 1:

Γράψτε μία συνάρτηση με όνομα `favorite_movie` η οποία παίρνει σαν είσοδο μία συμβολοσειρά (`movie_title`).

Αν η συμβολοσειρά περιέχει τη λέξη "Batman" η συνάρτηση να τυπώνει "good choice" αλλιώς να τυπώνει "awful taste"
Ελέγξτε στο πρόγραμμά σας ότι η συνάρτηση λειτουργεί σωστά.

Άσκηση 2:

Γράψτε μία συνάρτηση με όνομα `favorite_author` η οποία παίρνει σαν είσοδο μία συμβολοσειρά (`author`).

Αν η συμβολοσειρά περιέχει τη λέξη "Tolkien" η συνάρτηση να τυπώνει 500 φορές "Tolkien is the best" αλλιώς να τυπώνει το όνομα του συγγραφέα και έπειτα "is good." μία φορά.
Ελέγξτε στο πρόγραμμά σας ότι η συνάρτηση λειτουργεί σωστά.

Μία επιστρεφόμενη τιμή

- Χρησιμοποιούμε τη return ακολουθούμενη από την τιμή που θέλουμε να επιστρέψει η συνάρτηση.
- Αποθηκεύουμε την επιστρεφόμενη τιμή σε κάποια μεταβλητή καταχωρώντας σε αυτήν το αποτέλεσμα της κλήσης της συνάρτησης.

```
def func_name(param1, param2,...):
    ...
    ...
    return value

ret_value = func_name(arg1, arg2,...)
```

Παρατήρηση:

Η return προκαλεί άμεσο τερματισμό της εκτέλεσης της συνάρτησης (δεν συνεχίζει στις επόμενες εντολές)

Παράδειγμα 3: return.value.py

```
def is_odd(number):
    if number % 2 == 1:
        return True
    return False

print(f"2 is odd: {is_odd(2)}")
```

Πολλές επιστρεφόμενες τιμές

- Χρησιμοποιούμε τη return ακολουθούμενη από τις τιμές, χωρισμένες με κόμματα
- Αποθηκεύουμε τις επιστρεφόμενες τιμές, χρησιμοποιώντας αντίστοιχα μεταβλητές χωρισμένες με κόμματα.

```
def func_name(param1, param2,...):
    ...
    ...
    return value1, value2

ret_value1, ret_value2 = func_name(arg1, arg2,...)
```

Παράδειγμα 4: return.values.py

```
def square_cube(number):
    return number ** 2, number ** 3

num = 5
square, cube = square_cube(num)
print(f"{num}^2={square}, {num}^3={cube}")
```

Παρατήρηση:

Μπορούμε να χρησιμοποιήσουμε μια “ξερή” return σε συνάρτηση που δεν επιστρέφει τίποτα. Προκαλεί άμεσο τερματισμό της συνάρτησης και επιστρέφει την ειδική τιμή “None”

```
# none.py
def f():
    return

ret = f()
print(f"{ret}, {type(ret)}")
```

Άσκηση 3: Έλεγχος ακεραίου

Κατασκευάστε μία συνάρτηση με όνομα `input_integer`

- Δεν θα παίρνει ορίσματα
- Θα προτρέπει το χρήστη να εισάγει έναν ακέραιο
- Θα ελέγχει ότι η είσοδος του χρήστη δεν περιέχει μη έγκυρους χαρακτήρες (π.χ. γράμματα). Αν η είσοδος είναι λάθος θα βγάζει κατάλληλο μήνυμα και ο χρήστης θα εισάγει εκ νέου την είσοδο.
- Θα επιστρέφει τον ακέραιο που διάβασε.

Ελέγξτε ότι η συνάρτησή σας λειτουργεί σωστά.

Άσκηση 4: Έλεγχος πραγματικού

Κατασκευάστε μία συνάρτηση με όνομα `input_float`

- Δεν θα παίρνει ορίσματα
- Θα προτρέπει το χρήστη να εισάγει έναν πραγματικό
- Ο χρήστης θα πρέπει να εισάγει είτε έναν ακέραιο (π.χ. το 2) ο οποίος θα πρέπει να αποθηκευτεί ως πραγματικός, είτε έναν πραγματικό (π.χ. το 2.1). Να γίνονται όλοι οι απαραίτητοι έλεγχοι στην είσοδο (όχι γράμματα, σύμβολα κ.λπ). Να επαναλαμβάνει αν ο χρήστης έβαλε λάθος είσοδο
- Θα επιστρέφει τον πραγματικό που διάβασε.

Ελέγξτε ότι η συνάρτησή σας λειτουργεί σωστά.

Άσκηση 5: Μια βιβλιοθήκη μελέτης αριθμών

Κατασκευάστε τις συναρτήσεις:

- `is_odd(number)`: Περιττός
- `is_even(number)`: Άρτιος
- `is_prime(number)`: Πρώτος
- `is_square(number)`: Τετράγωνο (αριθμός που προκύπτει αν υψώσουμε έναν φυσικό στο τετράγωνο)
- `is_cube(number)`: Κύβος (αριθμός που προκύπτει αν υψώσουμε έναν φυσικό στην 3η δύναμη)

Ελέγξτε αν οι συναρτήσεις λειτουργούν σωστά ως εξής:

Τυπώστε τους φυσικούς από το 1 έως το 100 και τυπώστε για τον καθένα από αυτούς, ποιες από τις παραπάνω ιδιότητες ικανοποιούν.

[Υπενθύμιση: Ασχοληθήκαμε με τον έλεγχο για το αν ένας αριθμός είναι πρώτος, στο μάθημα 6, άσκηση 4]

Άσκηση 6: Γεωμετρία

Κατασκευάστε μια βιβλιοθήκη συναρτήσεων για τον υπολογισμό τύπων γεωμετρικών σχημάτων.

- `triangle_area` (Εμβαδόν τριγώνου = $\frac{\beta \cdot \upsilon}{2}$)
- `square_perimeter` (Περίμετρος τετραγώνου = 4α)
- `square_area` (Εμβαδόν τετραγώνου = α^2)
- `circle_perimeter` (Περίμετρος Κύκλου = $2\pi R$)
- `circle_area` (Εμβαδόν κύκλου = πR^2)

Ελέγξτε ότι οι συναρτήσεις σας λειτουργούν σωστά.

Hint: Ορίστε μια σταθερά `PI` στην αρχή του προγράμματος με την τιμή: 3.14159265359

Άσκηση 7: Εκτύπωση vs Επιστροφή

Κατασκευάστε τη συνάρτηση:

- `digits_print(number)`: Παίρνει σαν παράμετρο έναν τριψήφιο ακέραιο και να εκτυπώνει τα τρία ψηφία διαχωρίζοντας τα.
- Π.χ. η τελική εκτύπωση για το 352 να είναι:

```
1st digit: 3
2nd digit: 5
3rd digit: 2
```

Επαναλάβετε την άσκηση κατασκευάζοντας τη συνάρτηση `digits(number)`, η οποία επιστρέφει τα τρία ψηφία ή `None` (σε περίπτωση λάθους εισόδου).

Επαληθεύστε τη σωστή λειτουργία των δύο συναρτήσεων.

- Έχουμε πρόσβαση σε κάθε μεταβλητή που έχει δηλωθεί πριν την κλήση μίας συνάρτησης, μέσα από τη συνάρτηση (global1.py)

```
def f():  
    print(x)  
  
x = 5  
f()
```

- Για το λόγο αυτό, οι μεταβλητές του κυρίως προγράμματος λέγονται και **καθολικές μεταβλητές**
- Αντίθετα οι μεταβλητές που έχουμε δηλώσει στο σώμα μίας συνάρτησης λέγονται **τοπικές μεταβλητές**. (local1.py)

```
def f():  
    x = 5  
    print(x)  
  
f()
```

- Μία τοπική μεταβλητή, ωστόσο, δεν είναι προσβάσιμη έξω από τη συνάρτηση. Το ακόλουθο πρόγραμμα θα “χτυπήσει”

```
def f():  
    x = 5  
  
f()  
print(x)
```

- Μία μεταβλητή δεν μπορεί να είναι ταυτόχρονα καθολική και τοπική. Αν στον κώδικα της συνάρτησης ορίζεται η τιμή ή αλλάζει η τιμή σε μία μεταβλητή, αυτή είναι αυτόματα τοπική.
- Έτσι στο ακόλουθο πρόγραμμα, οι δύο μεταβλητές x είναι διαφορετικές (μία τοπική στη συνάρτηση και μία καθολική και στην συνάρτηση είναι ορατή μόνο η τοπική μεταβλητή) (global.vs.local.py)

```
def f():  
    x = 3  
    print(x)  
  
x = 2  
f()  
print(x)
```

- Ωστόσο υπάρχει η δυνατότητα να αναφερόμαστε στην καθολική μεταβλητή μέσα στη συνάρτηση, βάζοντας τη λέξη κλειδί global ακολουθούμενη από το όνομα της μεταβλητής (global2.py):

```
def f():  
    global x  
    x = 3  
    print(x)  
  
x = 2  
f()  
print(x)
```


Παρατήρηση:

- Γενικά η χρήση καθολικών μεταβλητών θεωρείται κακή προγραμματιστική πρακτική σε μεγάλα προγράμματα και θα πρέπει να αποφεύγεται (οδηγεί συχνά σε λάθη)
- Θα δούμε (επόμενη διαφάνεια) έναν τρόπο για να αποφεύγουμε πλήρως τις καθολικές μεταβλητές.
- Θα χρησιμοποιούμε καθολικές μεταβλητές, μόνο σε πολύ ειδικές περιπτώσεις, για πράγματα που είναι σίγουρο ότι δεν πρόκειται να γίνει κακή προγραμματιστική χρήση.

Άσκηση 8.1: Ανισόρροπο παιχνίδι τράπουλας

Ξεκινήστε την κατασκευή, παίρνοντας την κατασκευή του deck από την άσκηση 5 του μαθήματος 7 “Σύνολα”. Το deck να ορίζεται ως καθολική μεταβλητή.

Άσκηση 8.2: Συνέχεια

Σε αυτό το ιδιότυπο παιχνίδι 2 παίκτες πετούν την τράπουλα ψηλά στον αέρα σκορπίζοντας τα χαρτιά. Έπειτα οι δύο παίκτες μαζεύουν όλα τα χαρτιά. Νικάει ο παίκτης που μάζεψε τα περισσότερα χαρτιά.

- Ορίστε δύο ακόμη καθολικές μεταβλητές με ονόματα player1 και player2. Αυτές θα αναπαριστούν τα χαρτιά που έχει μαζέψει κάθε παίκτης.
- Ορίστε δύο συναρτήσεις με ονόματα player1_pick και player2_pick. Οι συναρτήσεις θα επιλέγουν ένα τυχαίο χαρτί από το deck και θα το τοποθετούν στο αντίστοιχο χέρι των παικτών.
- Ορίστε μία συνάρτηση με όνομα play.
 - Να τυπώνεται κάποιο εισαγωγικό μήνυμα
 - Επαναληπτικά, με τυχαίο τρόπο, θα επιλέγεται ένας παίκτης και θα επιλέγεται ένα χαρτί γι’ αυτόν, εφόσπου εξαντληθεί το deck
 - Θα εκτυπώνεται ποιος νίκησε και πόσα χαρτιά μάζεψε κάθε παίκτης.

- Για να απομονώσουμε τις ενέργειες που γίνονται στο πρόγραμμα από τις ενέργειες που γίνονται στις συναρτήσεις και να μην έχουμε συγκρούσεις ονομάτων, κάνουμε το εξής:
- Ορίζουμε μία ειδική συνάρτηση, τη main, στην οποία περικλείουμε τις ενέργειες που θέλουμε να γίνονται στο πρόγραμμα:
- Π.χ. το ακόλουθο πρόγραμμα:

```
def f():
    print("something")

def g():
    print("something else")

f()
g()
```

- Αναπροσαρμόζεται ως εξής:

```
def f():
    print("something")

def g():
    print("something else")

def main():
    f()
    g()

main()
```

Με βάση τα προηγούμενα η συνιστώμενη δομή ενός (μεγάλου) προγράμματος είναι η εξής:

- Δήλωση σταθερών
- Δήλωση καθολικών μεταβλητών
- Δήλωση συναρτήσεων
- Συνάρτηση main

```
# constants
PI = 3.14

# globals
deck = {"ace", "spade"}

# functions
def f():
    print("something")

def g():
    print("something else")

# main
def main():
    f()
    g()

main()
```

Άσκηση 9: Τρίλιζα (ξανά)

Στο Μάθημα 7, άσκηση 6, υλοποιήσαμε τη τρίλιζα. Τώρα μπορούμε να κάνουμε τον κώδικά της πολύ κομψότερο, ορίζοντας κατάλληλες συναρτήσεις:

- Μοναδική καθολική μεταβλητή θα είναι το board
- Ορίστε τη συνάρτηση `print_board()` η οποία θα εκτυπώνει το τρέχον board.
- Σκεφθείτε πως μπορείτε να “συμμαζέψετε” τον έλεγχο νίκης για να είναι πιο κομψός με χρήση συναρτήσεων
- Να υπάρχει συνάρτηση `main`

Ορίσματα και διατήρηση αλλαγών σε αυτά μετά από την κλήση:

- Κανόνας 1: Τα immutable ορίσματα δεν διατηρούν τις αλλαγές που γίνονται σε αυτά εκτός της συνάρτησης

```
# immutable.arguments.py
def f(arg):
    print(arg)
    arg = "Change!"
    print(arg)

s = "Initial"
print(s)
f(s)
print(s)
```

- Κανόνας 2: Τα mutable ορίσματα διατηρούν τις αλλαγές που γίνονται σε αυτά εκτός της συνάρτησης, εφόσον αυτές δεν είναι καταχωρήσεις (mutable.arguments.py)

```
def f(arg):
    print(arg)
    arg.append(3)
    print(arg)

l = [1,2]
print(l)
f(l)
print(l)
```

- Κανόνας 3: Τα mutable ορίσματα δεν διατηρούν τις αλλαγές που γίνονται σε αυτά εκτός της συνάρτησης, εφόσον αυτές έπονται μίας καταχώρησης (mutable.arguments.assignment.py)

```
def f(arg):
    print(arg)
    arg = [3]
    print(arg)

l = [1,2]
print(l)
f(l)
print(l)
```

Παρατήρηση:

- Το πέρασμα των παραμέτρων λέμε ότι γίνεται μέσω αναφοράς αντικειμένου (call by object reference) και ακολουθεί τους κανόνες που μελετήσαμε.
- Προσοχή ότι δεν υπάρχει ο διαχωρισμός για να επιλέγουμε προγραμματιστικά αν θα αλλάζει η τιμή ή όχι ενός ορίσματος από την κλήση της συνάρτησης (όπως π.χ. σε άλλες γλώσσες προγραμματισμού έχουμε την κλήση μέσω αναφοράς ή μέσω τιμής (call by value, call by reference))

Άσκηση 10: id αναφορών

Εκτελέστε εκ νέου τα προγράμματα της προηγούμενης διαφάνειας, ενσωματώνοντας κατάλληλες εκτυπώσεις του id των εμπλεκόμενων αντικειμένων για να διαπιστώστε σε ποια αντικείμενα γίνονται πραγματικά οι αλλαγές κατά τη κλήση της συνάρτησης.

Άσκηση 11: Και πως θα αλλάξουμε ένα immutable;

Για κάποιους λόγους σε κάποιο πρόγραμμα θέλουμε:

- Να έχουμε τη main να ορίζει μια ακέραια μεταβλητή.
- Αυτή η μεταβλητή να διοχετεύεται σε μια συνάρτηση, η οποία επιθυμούμε να διπλασιάζει τη τιμή της.
- Επιθυμούμε όμως αυτή η αλλαγή να διατηρείται μετά τη κλήση της συνάρτησης.
- Πως μπορούμε να το καταφέρουμε;

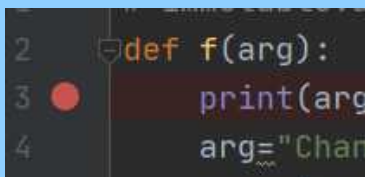
Σημείωση: Η άσκηση αυτή προσομοιώνει τη λειτουργία “call by reference” που έχουμε σε άλλες γλώσσες προγραμματισμού, στην οποία η αλλαγή στην τιμή του ορίσματος διατηρείται και μετά το πέρας της συνάρτησης.

Κάθε IDE όπως το PyCharm ενσωματώνει ένα μηχανισμό debugging

- Με αυτόν μπορούμε να κάνουμε παύση στην εκτέλεση του προγράμματος
- και έπειτα να τρέχουμε βήμα – βήμα τον κώδικα παρακολουθώντας τις τιμές των μεταβλητών

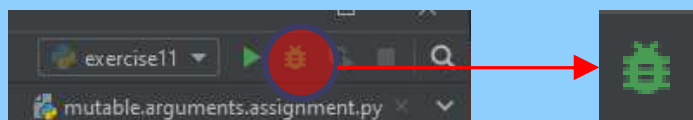
Debugging με το Pycharm

- **Βήμα 1:** Βάζουμε ένα breakpoint (σημείο παύσης) στα σημεία του προγράμματος που θέλουμε να ελέγχουμε:



```
2 def f(arg):  
3     print(arg)  
4     arg="Char"
```

- **Βήμα 2:** Πατάμε το ζουζούνι (αντί για το play) πάνω δεξιά:



- **Βήμα 3:** Στο κάτω μέρος της οθόνης ανοίγει ένα μενού με επιλογές (δίπλα στην κονσόλα):



- και πλέον με τα κουμπάκια μπορούμε να τρέχουμε τον κώδικα βήμα – βήμα, να βλέπουμε την εξέλιξη των μεταβλητών, κ.ο.κ. (βλέπε βίντεο για περαιτέρω επεξήγηση)

Άσκηση 12: Ο προγραμματιστής θέλει να διαβάσει 3 ακέραιους από την είσοδο, να βρει το μεγαλύτερο από αυτούς, έστω n και να τυπώσει τα τετράγωνα όλων των αριθμών από το 1 έως το n . Ωστόσο το πρόγραμμα έβγαλε λάθος. Βοηθήστε τον προγραμματιστή να βρεί τα λάθη (exercise12_initial.py)

```
def max3(x, y, z):  
    if x > y and x > z:  
        return x  
    elif y > x and y > z:  
        return y  
    else:  
        return z  
  
x = input("Give a number: ")  
y = input("Give a number: ")  
z = input("Give a number: ")  
  
n = max3(x, y, z)  
  
for i in range(n):  
    sq = i*i  
    print(sq)
```

Άσκηση 13.1: Black Jack

Στο Black Jack (απλούστευση του πραγματικού):

- Παίζει πρώτα ο παίκτης. Στόχος του είναι να φτάσει σε άθροισμα χαρτιών ίσο με 21 (ο άσσος μετράει για 1 ή 11 (επιλογή του παίκτη) και οι φιγούρες μετράνε για 10)
 - Αρχικά του δίνονται δύο χαρτιά.
 - Ο παίκτης επαναληπτικά επιλέγει αν θα τραβήξει νέο χαρτί (για να φτάσει σε άθροισμα πιο κοντά σε 21) ή θα σταματήσει στο άθροισμα των χαρτιών που έχει ήδη τραβήξει.
 - Αν το άθροισμα είναι 21, τότε ο παίκτης κερδίζει αμέσως.
 - Αν το αθροισμα είναι πάνω από 21, τότε ο παίκτης χάνει αμέσως
- Έπειτα παίζει ο υπολογιστής. Γνωρίζει το άθροισμα των χαρτιών του παίκτη και τραβάει συνεχώς χαρτιά εως ότου:
 - Είτε βγάλει άθροισμα πάνω από 21, οπότε χάνει.
 - Είτε βγάλει άθροισμα μεγαλύτερο ή ίσο με του παίκτη, οπότε κερδίζει.

Ξεκινήστε την κατασκευή, παίρνοντας την κατασκευή του deck από την άσκηση 5 του μαθήματος 7 “Σύνολα”. Το deck να είναι η μοναδική καθολική μεταβλητή.

Έπειτα κατασκευάστε μία συνάρτηση (hand_value) η οποία μετράει το «άθροισμα» ενός συνόλου χαρτιών.

Άσκηση 13.2: Παίκτης

- Υλοποιήστε σε μία συνάρτηση τη λογική του παίκτη (player). Να επιστρέφει το άθροισμα των αριθμών των χαρτιών του.
- Να εκτυπώνεται στη συνάρτηση και το τελικό χέρι του παίκτη.

Άσκηση 13.3: Υπολογιστής

- Στη main να ελέγχεται αν ο παίκτης έχασε.
- Αν όχι να παίζει ο υπολογιστής.
- Υλοποιήστε πάλι σε μία συνάρτηση τη λογική του υπολογιστή (computer). Να παίρνει ως παράμετρο το άθροισμα του παίκτη και να επιστρέφει το άθροισμα των αριθμών των χαρτιών του υπολογιστή.
- Να εκτυπώνεται στη main το τελικό αποτέλεσμα.

Άσκηση 13.4: Γύροι - Σκορ

Υλοποιήστε έναν μηχανισμό γύρων. Ο παίκτης, μετά από κάθε γύρο, να επιλέγει αν θέλει να συνεχίσει ή όχι και να τυπώνεται ο τρέχον γύρος και το τρέχον συνολικό σκορ.

[Θεωρείστε ότι σε κάθε γύρο το deck πρέπει να αρχικοποιείται, δηλαδή να ξαναπαίνουν σε αυτό τα χαρτιά που είχαν χρησιμοποιηθεί στον προηγούμενο γύρο]

Άσκηση 14.1: Αναδόμηση του προγράμματος

Αναδομήστε το πρόγραμμα του μαθήματος 10 – άσκηση 8, ώστε να χρησιμοποιεί τις συναρτήσεις:

- **main()**
- **print_pupil(pupil):** Εκτυπώνει μορφοποιημένα το μαθητή.
- **create_pupil():** Δημιουργεί ένα μαθητή και ενημερώνει τη λίστα μαθητών.
- **next_id():** Επιστρέφει το επόμενο διαθέσιμο id

Μόνο η λίστα μαθητών επιτρέπεται να δηλωθεί ως καθολική μεταβλητή.

Άσκηση 14.2: Εκτύπωση

Κατασκευάστε ένα υπό-μενού στην επιλογή “2” με τις εξής επιλογές:

1. Εκτύπωση Μαθητή
2. Εκτύπωση όλων των μαθητών (αναλυτική)
3. Εκτύπωση όλων των μαθητών (μόνο ονόματα)

Υλοποιήστε τη λειτουργικότητα ώστε να βγαίνει μήνυμα λάθους σε περίπτωση λάθους εισόδου.

Άσκηση 14.3: Υλοποίηση Επιλογών

Συγκεκριμένα:

- Στην εκτύπωση μαθητή, να ζητείται από το χρήστη το id και να εκτυπώνεται ο μαθητής.
- Στην αναλυτική εκτύπωση όλων των μαθητών, να κατάσκευάσετε νέα συνάρτηση με όνομα `print_pupils_details` στην οποία να πραγματοποιείται εκτύπωση κάθε μαθητή με τη κλήση της συνάρτησης `print_pupil`
- Στην εκτύπωση των ονομάτων, να καλείται νέα συνάρτηση με όνομα `print_pupils_names` η οποία να εκτυπώνει για κάθε μαθητή, το όνομα, το πρώτο γράμμα του ονόματος πατρός και το επώνυμο.

“Special cases aren't special enough to break the rules.”

Zen of Python #8