Συναρτήσεις

Θεωρία & Ασκήσεις

Επαναχρησιμοποίηση κώδικα!

- Σημαντική τεχνική σε οποιαδήποτε γλώσσα προγραμματισμού
- Λόγος: Στους κώδικες η αύξηση του μεγέθους οδηγεί στην δυσκολία συντήρηση τους.
- Στους μεγάλους κώδικες ισχύει η αρχή του **Don't Repeat Yourself (DRY)**. *Θυμίζει κάτι;*
- Οι κακοί και επαναληπτικοί κώδικες υπάγονται στην αρχή **WET (Write Everything Twice** ή **We Enjoy Typing)**

Συναρτήσεις

- Πολλές συναρτήσεις έχουν ήδη χρησιμοποιηθεί σε παλαιότερα μαθήματα.
- Γενική μορφή: *όνομα*(*ορίσματα*).
- Σε οποιαδήποτε γραμμή κώδικα εμφανίζεται αυτή η μορφή, λέμε ότι έχουμε **κλήση συνάρτησης**.
- π.χ.
  - print(“hello world”)
  - range(2, 20)
  - x = str(12)
  - range(10, 20, 3)

Τύποι συναρτήσεων

- Στην γλώσσα Python εμφανίζονται **2** τύποι συναρτήσεων
  - Προκαθορισμένες
  - Χρήστη

- Οι προκαθορισμένες (*built-in*) αποτελούν κομμάτι της γλώσσας.

- Οι χρήστη (*user defined*) αποτελούν συναρτήσεις που δημιουργούνται από τον χρήστη μέσω της δεσμευμένης λέξης **def**.

Άτυπες κατηγορίες

- Όπως προαναφέρθηκε, πέραν των προκαθορισμένων συναρτήσεων, ένας χρήστης μπορεί να δημιουργήσει την δική του συλλογή.

- Άτυπες κατηγορίες συναρτήσεων (με βάση το αποτέλεσμα τους)
  - Απλές (*simple*): *Συναρτήσεις που επιστρέφουν κάποια τιμή.*
  - Κενές (*void*): *Συναρτήσεις που δεν επιστρέφουν κάποια τιμή.*

- Άτυπες κατηγορίες συναρτήσεων (με βάση τα ορίσματα)
  - Με ορίσματα.
  - Χωρίς ορίσματα.

Δημιουργία συναρτήσεων

- Παράδειγμα κώδικα που δημιουργεί μια συνάρτηση με το όνομα func().

```
1  def my_func():
2      print("Spam");
3      print("Spam");
4      print("Spam");
5
6  print("We call the function")
7  my_func()
8  print("\nEND OF CODE")
9
```

Δημιουργία συναρτήσεων (συνέχεια)

- Το κομμάτι κώδικα που βρίσκεται μέσα στην συνάρτηση εκτελείται μόνο όταν γίνει κλήση της συνάρτησης αυτής.

- Κάθε κομμάτι κώδικα (*block*) ξεκινά με τον χαρακτήρα “ **:** ” και γράφεται σε εσοχή (*tab*).

- *Που αλλού έχουμε δει αυτήν την σύνταξη;*
- *Θα εκτελεστεί σωστά ο κώδικας χωρίς εσοχές;*

- Οι συναρτήσεις πρέπει **πάντα** να ορίζονται/δηλώνονται πρίν να κληθούν.
  - *Τί θα γίνει αν δεν ορίσω μια συνάρτηση πρίν την κλήση της;*

Δημιουργία συναρτήσεων (συνέχεια)

- Μέχρι τώρα είδαμε συναρτήσεις δίχως ορίσματα. Όμως οι περισσότερες συναρτήσεις εμφανίζουν ένα ή και περισσότερα.

- Ας εξετάσουμε αυτήν την περίπτωση μέσω παραδείγματος:

```
1  def print_with_wow(word):
2      print(word + "!")
3
4  print_with_wow("spam")
5  print_with_wow("neo")
6  print_with_wow("python")
7
```

Δημιουργία συναρτήσεων (συνέχεια)

- Γενικά μπορώ να χρησιμοποιήσω περισσότερα από ένα ορίσματα, τα οποία χωρίζονται με κόμματα.

```
1  def print_sum_twice(x, y):
2      print(x + y)
3      print(x + y)
4
5  print_sum_twice(5, 8)
6
```

Τοπικότητα αναφορών

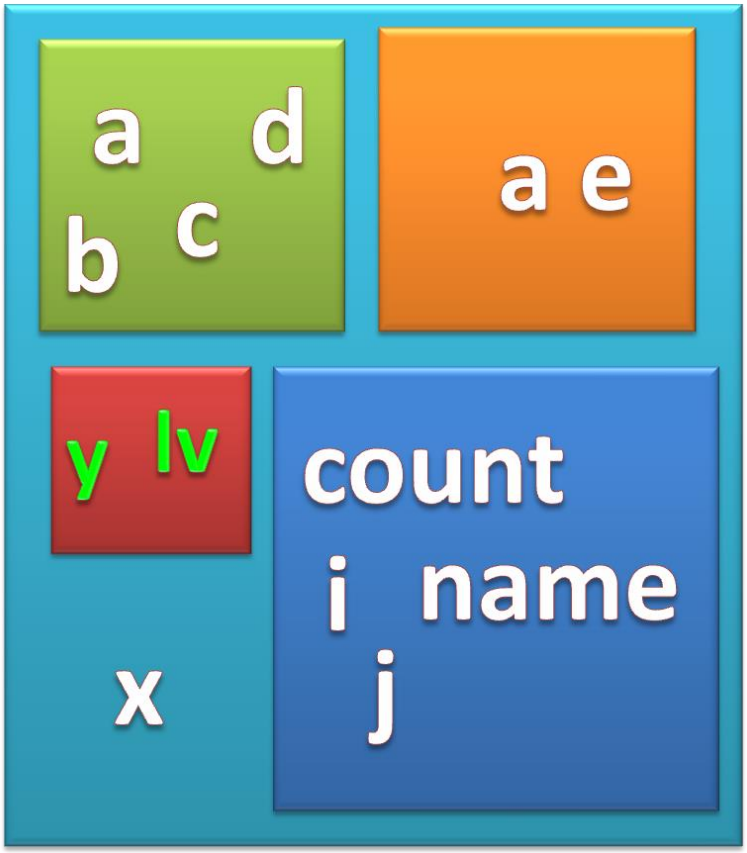
- Τα ορίσματα των συναρτήσεων μπορούν να χρησιμοποιηθούν σαν μεταβλητές μέσα στον κώδικα της συνάρτησης ώς παράμετροι.

- Παρόλα αυτά δεν μπορεί να γίνει αναφορά στις παραμέτρους εκτός του κώδικα της συνάρτησης.

- Τοπικότητα αναφορών: ορίζει ποιές μεταβλητές «φαίνονται» στα κομμάτια κώδικα ενός προγράμματος.
  - Καθολικές (*global*): *μεταβλητές που «φαίνονται» σε όλα τα κομμάτια κώδικα.*
  - Τοπικές (*local*): *μεταβλητές που «φαίνονται» μόνο στο κομμάτι κώδικα που τις δημιούργησε.*

Τοπικότητα αναφορών (συνέχεια)

- Περιγράψτε την τοπικότητα των παρακάτω μεταβλητών:



Τοπικότητα αναφορών (συνέχεια)

- Τεχνικά, οι παράμετροι είναι οι μεταβλητές κατά την δήλωση μιας συνάρτησης. Τα ορίσματα είναι οι τιμές που θα «μπουν» στις παραμέτρους κατά την κλήση μιας συνάρτησης.

```
1  def args(a, b, c, d):
2      sum = a + b + c + d
3      print(sum)
4
5  x = 10
6  y = 20
7  z = 30
8  w = 40
9  args(x, y, z, w)
10
```

Επιστροφή τιμής

- Διάφορες συναρτήσεις όπως οι int() και η str() επιστρέφουν μια τιμή, η οποία μπορεί να χρησιμοποιηθεί αργότερα κατά την διάρκεια του κώδικα.

- Για να το επιτύχουμε αυτό, χρησιμοποιούμε στις συναρτήσεις μας την εντολή **return**.

- Μόλις επιστραφεί μια τιμή από την συνάρτηση σταματά η εκτέλεση. Οποιοδήποτε κομμάτι κώδικα μετά το **return** δεν θα εκτελεστεί ποτέ. *Τι μας θυμίζει αυτό;*

Επιστροφή τιμής (συνέχεια)

- Δύο σύντομα παραδείγματα με επιστροφή τιμής

```
1  def sum2(a, b):
2      sum = a + b
3      return sum
4      print(sum)
5
6  x = sum2(5, 6)
7  print(x)
8
9  y = 12
10 z = 13
11 print(sum2(y, z))
12
```

```
1  def maximum(a, b):
2      if a > b:
3          return a
4      else:
5          return b
6
7  print(maximum(4, 7))
8
9  z = maximum(9, 9.5)
10 print(z)
11
```

Σχόλια

- Για την διευκόλυνση κατανόησης κώδικα, οι προγραμματιστές χρησιμοποιούν **σχόλια** *(comments)*.

- Ιδιότητες:
  - Τα σχόλια δεν επηρεάζουν την εκτέλεση του κώδικα
  - Στην Python σχόλια δημιουργούνται με την εισαγωγή της δίεσης στην αρχή της γραμμής.
    - Π.χ. # this code is not going to work :)
  - Από την στιγμή που εντοπιστεί η δίεση, οποιοδήποτε κείμενο μετά από αυτήν αγνοείται από την εκτέλεση.

Σχόλια (συνέχεια)

- Παράδειγμα

```
1  # comments are like that
2  # begin the program
3
4  x = 10
5  print(x)
6
7  while x > 0:
8      print(x)
9      x -= 1        # decrease x
10
11 # end of program
12
```

Docstrings

- Παρόμοια λειτουργία με τα σχόλια.
- Είναι πιο ειδικά και έχουν διαφορετική σύνταξη.
- Εισάγονται με την χρήση τριών εισαγωγικών. Ουσιαστικά είναι ένα αλφαριθμητικό που «πιάνει» πάνω από μια γραμμή.

Docstrings (συνέχεια)

- Παράδειγμα

```
""" Example of Docstrings
    we begin the program """
x = 10
print(x)

"""  now we will sit back
    and watch some loops!! """

while x > 0:
    print(x)
    x -= 1  #   decrease by 1

#   end of program

```

Συναρτήσεις ως «μεταβλητές»

- Παρότι δημιουργούνται διαφορετικά από τις κανονικές μεταβλητές, οι συναρτήσεις συμπεριφέρονται όπως κάθε άλλο είδος τιμής.

- Μπορώ να κάνω αναθέσεις με το όνομα της
- Μπορώ να περάσω συναρτήσεις ως ορίσματα σε άλλες συναρτήσεις

- Αν και την χαρακτηρίσαμε έτσι, οι συναρτήσεις δεν θεωρούνται μεταβλητές, αλλά αντικείμενα. Για αντικείμενα θα μιλήσουμε σε άλλη παράδοση.

- Οι συναρτήσεις οι οποίες καλούν άλλες συναρτήσεις ονομάζονται **συναρτήσεις υψηλής τάξης (*High-Order Functions)***

Συναρτήσεις ως «μεταβλητές» (συνέχεια)

Ανάθεση τιμής

```
1  def multiply(x, y):
2      return x*y
3
4  a = 4
5  b = 3
6  operation = multiply
7  print(operation(a, b))
8
```

Πέρασμα ως όρισμα

```
1  def sum2(a, b):
2      return a + b
3
4  def do_twice(func, x, y):
5      return func(func(x,y), func(x,y))
6
7  a = 5
8  b = 6
9
10 print(do_twice(sum2, a, b))
11
```

Συναρτησιακός Προγραμματισμός

Βασικές έννοιες και τεχνικές

Συναρτησιακός Προγραμματισμός

- Όπως γίνεται αντιληπτό, η έννοια Συναρτησιακός Προγραμματισμός (*Functional Programming*) αφορά το στύλ προγραμματισμού που έχει ως βάση τις συναρτήσεις.

- Σημαντικό κομμάτι του συναρτησιακού προγραμματισμού είναι η χρήση **συναρτήσεων υψηλής τάξης**.

- Κύριος στόχος του Σ.Π. Είναι να αναζητεί αγνές συναρτήσεις (*pure functions*).

- Αγνές ονομάζονται οι συναρτήσεις που δεν αλληλεπιδρούν άμεσα με τον κύριο κώδικα, και των οποίων η τιμή επιστροφής βασίζεται εξ'ολοκλήρου στα ορίσματα της.

Διαφορές αγνών και μη-αγνών

Αγνή Συνάρτηση (Pure)

```
1  # pure function
2  def pure_function(x, y):
3      temp = x + 2*y
4      return temp/(2*x + y)
5
```

Μη-αγνή Συνάρτηση (Impure)

```
1  # impure function
2
3  some_list = []
4
5  def impure_function(arg):
6      some_list.append(arg)
7
8
```

Πλεονεκτήματα & Μειονεκτήματα

- Η χρήση των αγνών συναρτήσεων έχει και τα θετικά, αλλά και τα αρνητικά
- Θετικά:
  - Οι αγνές συναρτήσεις είναι εύκολες στην κατανόηση και τον έλεγχο.
  - Είναι πιο αποτελεσματικές, καθώς υπάρχει η μέθοδος της απομνημόνευσης για επιτάχυνση του προγράμματος (*memoization).*
- Αρνητικά:
  - Περιπλέκουν την απλή διαδικασία εισόδου-εξόδου.
  - Σε μερικές περιπτώσεις μπορεί να γίνει δύσκολη η σύλληψη και η υλοποίηση τους.

Αναδρομή

- Η αναδρομή αποτελεί σημαντική τεχνική στον συναρτησιακό προγραμματισμό. Το βασικό στοιχείο της αναδρομής είναι οτι η συνάρτηση που την εκτελεί καλεί τον εαυτό της μέσα στο σώμα της.

- Χρησιμοποιείται για την διαίρεση ενός προβλήματος σε ευκολότερα υποπροβλήματα.

- Για να πετύχει μια αναδρομή πρέπει να λάβουμε υπόψην τα παρακάτω:
  - Μια βασική περίπτωση (*base case*), κατά την οποία θα «σπάσει» η αναδρομή.
  - Με ποιό τρόπο θέλουμε να διαιρέσουμε το κυρίος πρόβλημα μας σε υποπροβλήματα.

Αναδρομή - Παραγοντικό

- Στα μαθηματικά το **παραγοντικό** ενός φυσικού αριθμού ν συμβολίζεται με ν!, διαβάζεται *νι παραγοντικό*, και είναι το γινόμενο όλων των θετικών ακεραίων μικρότερων ή ίσων με ν.

- $\boldsymbol{\nu! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot \nu}$

- Βασική ιδιότητα: $\nu! = (\nu - 1)! \cdot \nu$

- Παραδείγματα:
  - $5! = 4! \cdot 5 = 3! \cdot 4 \cdot 5 = \ldots = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 =$ ***120***
  - $19! = 18! \cdot 19 = 17! \cdot 18 \cdot 19 = \ldots = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \ldots 18 \cdot 19 =$ ***121645100408832000***

Αναδρομή - Κώδικας Παραγοντικού

```
1  # Factorial Calculation
2
3  def factorial(num):
4
5      if num == 0:
6          return 1
7      else:
8          return num*factorial(num-1)
9
10
11 print(factorial(int(input("Give: "))))
12
```

Modules

Βασικές τεχνικές χρήσεις των Module

Modules

- Modules ή με την ακριβή ελληνική μετάφραση ***Ενότητες***, ονομάζουμε αρχεία python τα οποία είναι γραμμένα από τρίτους ή και από εμάς και εκτελούν κάποια εξειδικευμένη λειτουργία.

- Αποτελούν σημαντική βοήθεια για την χρήση διαφόρων λειτουργιών, απαλλάσοντας μας από την συγγραφή ήδη υπάρχοντος κώδικα.

- Η βασική χρήση ενός module γίνεται με την χρήση της εντολής **import module_name**.

- Το module χρησιμοποιείται μέσω του ονόματος που δώσαμε, του τελεστή τελεία, και του ονόματος της συνάρτησης που θέλουμε να καλέσουμε.

Modules (συνέχεια)

Όνομα Module	Λειτουργία
Random	Γεννήτρια τυχαίων συνόλων
Math	Μαθηματικές πράξεις

- Πέρα από τα παραπάνω βασικά modules, υπάρχουν πολλά περισσότερα στο δυαδίκτυο. Η ποικιλομορφία τους είναι πραγματικά εκπληκτική.

- Βασικοί τρόποι χρήσης ενός module είναι οι παρακάτω:
  - import *module_name*
  - from *module_name* import *element*
  - from *module_name* import *
  - import *module_name* as *user_defined_name*

Τρόποι χρήσης των Modules

import *module_name*

```
1  import math
2
3  a = math.pi
4  print(a)
5
6  b = math.pow(2, 3)
7  print(b)
8
```

from *module_name* import *element*

```
1  from math import pi
2
3  x = pi
4  print(x)
5
```

```
1  from math import pow
2
3  x = pow(2, 3)
4  print(x)
5
```

Τρόποι χρήσης των Modules

from *module_name* import *

```
1  from math import *
2
3  x = pow(2, 3)
4  print(x)
5
6  y = factorial(5)
7  print(y)
8
```

import *module_name* as *name*

```
1  import random as metal
2
3  a = metal.randint(2, 190)
4  print(a)
5
6  b = metal.sample(range(0,10), 5)
7  print(b)
8
```

```
1  import math as king
2
3  x = king.pi
4  print(x)
5
6  x = king.pow(5, 3)
7  print(x)
8
```

Το δικό μας Module!

- Φυσικά, μπορούμε να φτιάξουμε τα δικά μας modules με τις συναρτήσεις που εμείς δημιουργήσαμε.

- Για να γίνει import ενός δικού μας module, αρκεί το module να βρίσκεται στον ίδιο φάκελο με το αρχείο που το συμπεριλαμβάνει

Παράδειγμα

```
1  def print_capitals(name):
2      print(name.upper())
3
4  def print_smalls(name):
5      print(name.lower())
6
7  def name_letters(name):
8      return list(name)
9
10 def count_capitals(name):
11     cnt = 0
12     for i in name:
13         if i.isupper():
14             cnt += 1
15     return cnt
16
17 def count_lowercase(name):
18     cnt = 0
19     for i in name:
20         if i.islower():
21             cnt += 1
22     return cnt
23
24 c = 10.4
25
```

Παράδειγμα (συνέχεια)

```
1  import name_check as n
2
3  name = input("What's your name?\n")
4  n.print_capitals(name)
5
6  n.print_smalls(name)
7
8  nums = n.name_letters(name)
9  print(nums)
10
11 nums = n.count_capitals(name)
12 print(nums)
13
14 nums = n.count_lowercase(name)
15 print(nums)
16
17 print(n.c)
```

Ασκήσεις

Μερικές ασκησούλες για κατανόηση και εξάσκηση.

1. Να γραφεί συνάρτηση η οποία επιστρέφει τον μικρότερο μεταξύ
   α. Δύο αριθμών
   β. Τριών αριθμών

2. Ποιός είναι ο μεγαλύτερος αριθμός που εμφανίζεται στην έξοδο του παρακάτω κώδικα:

```
1  def print_nums(x):
2      for i in range(x):
3          print(i)
4          return
5
6  print_nums(10)
7
```

3. Να γραφεί συνάρτηση που υπολογίζει, εμφανίζει και επιστρέφει το άθροισμα των αριθμών από το 0 έως ένα arg, το οποίο θα δίνεται από τον χρήστη.

4. Ποιά είναι η έξοδος του παρακάτω κώδικα, και με ποιά αλλάγη θα μπορούσα να επιστρέψω: α) το διπλάσιο, β) το μισό.

```
1  def func(x):
2      res = 0
3      for i in range(x):
4          res += 1
5      return res
6
7  print(func(4))
8
```

5. Να γραφεί module το οποίο να περιέχει 3 διαφορετικές συναρτήσεις print, και μια μεταβλητή με το όνομα σας. Κατόπιν, να γραφει δεύτερο αρχείο python, το οποίο θα χρησιμοποιεί με έξυπνο τρόπο ότι περιέχει το module.

6. Έστω μια κλαδική συνάρτηση, όπως φαίνεται στην παρακάτω εικόνα. Να δημιουργήσετε module το οποίο περιλαμβάνει αυτή την συνάρτηση. Σε ένα δεύτερο αρχείο να δημιουργήσετε 2 λίστες: Η πρώτη θα περιέχει αρνητικές περιττές τιμές της συνάρτησης, ενώ η δεύτερη θα περιέχει άρτιες θετικές τιμές.

$$F(x) = \begin{cases} x^2, & x > 0 \\ 2 \cdot x + 0.5 \cdot x^3, & x < 0 \end{cases}$$

Πηγές

Το υλικό των διαφανειών βασίστηκε στις σειρές διαδικτυακών μαθημάτων των ιστοσελίδων: www.sololearn.com & www.tutorialspoint.com.

Όλοι οι κώδικες γράφηκαν στον κειμενογράφο notepad++.