Σημειώσεις

Προηγούμενο Επόμενο

Συναρτήσεις - ειδικότερα θέματα

Εμβέλεια μεταβλητών (τοπικές μεταβλητές)

Παράδειγμα. [Πηγή: Σημειώσεις Μ. Πλεξουσάκη.] Ας γράψουμε μια συνάρτηση η οποία, δεδομένου του φυσικού αριθμού n, υπολογίζει το άθροισμα $1^2 + 2^2 + \cdots + n^2$.

```
def sumsq(n):
    if n <= 0:
        return 0
    s = 0
    for i in range(1, n+1):
        s += i*i
    return s

print('Sum of the squares of the first four integers =', sumsq(4))
```

Το σώμα μιας συνάρτησης μπορεί να περιέχει οποιαδήποτε εντολή της Python, συμπεριλαμβανομένης, φυσικά, και της εντολής ανάθεσης. Το σώμα της παραπάνω συνάρτησης `sumsq` κάνει χρήση δύο μεταβλητών, των *i* και *s*. Η πρώτη είναι η μεταβλητή της ανακύκλωσης `for` και η δεύτερη περιέχει το τρέχων άθροισμα. Είναι σημαντικό να καταλάβουμε ότι αυτές οι μεταβλητές είναι "τοπικές" με την έννοια ότι είναι άγνωστες στο υπόλοιπο μέρος του προγράμματός μας. Αν, για παράδειγμα, προσπαθούσαμε να τυπώσουμε και την τιμή του μετρητή της ανακύκλωσης, εκτός του σώματος της συνάρτησης, θα παίρναμε το μήνυμα σφάλματος `NameError: name 'i' is not defined`. Το ίδιο ακριβώς συμβαίνει και με τα τυπικά ορίσματα μιας συνάρτησης: συμπεριφέρονται ως τοπικές μεταβλητές, είναι άγνωστες δηλαδή εκτός του σώματος της συνάρτησης.

Ας γράψουμε τον κώδικα για τη συνάρτηση `sumsq` με έναν διαφορετικό (και ισοδύναμο) τρόπο:

```
def sumsq(n):
```

```
    if n <= 0: return 0
    s = 0
    while n > 0:
        s += n*n
        n -= 1
    return s

print('Sum of the squares of the first four integers =', sumsq(4))
```

Επιπλέον, εκτός του σώματος της συνάρτησης `sumsq` μπορούμε να χρησιμοποιήσουμε το όνομα *n* ως όνομα μεταβλητής και η οποία δεν έχει καμμία απολύτως σχέση με το τυπικό όρισμα της `sumsq` με το ίδιο όνομα.

```
def sumsq(n):
    if n <= 0: return 0
    s = 0
    while n > 0:
        s += n*n
        n -= 1
    return s

n = 7
print('Sum of the squares of the first', n, 'integers =', sumsq(n))
```

Η τιμή της *n* εντός της συνάρτησης έχει αρχικά την τιμή 7 αλλά έχει την τιμή 0 όταν τελειώνει η εκτέλεση της συνάρτησης. Στο κύριο πρόγραμμα η τιμή της *n* είναι πάντα ίση με 7.

Παρατήρηση. Η Python, κατά τον ορισμό μιας συνάρτησης δημιουργεί τον λεγόμενο *χώρο ονομάτων* (namespace) ο οποίος περιέχει τόσο τα τυπικά ορίσματα της συνάρτησης αλλά και μεταβλητές τις οποίες χρησιμοποιεί, για τις ανάγκες της η συνάρτηση. Τα ονόματα αυτών των μεταβλητών είναι άγνωστα έξω από το χώρο ονομάτων. Αναφερόμαστε στις μεταβλητές που εμφανίζονται στο χώρο ονομάτων μιας συνάρτησης ως *τοπικές μεταβλητές*. Τα τυπικά ορίσματα συμπεριφέρονται κι αυτά ως τοπικές μεταβλητές μέσα στο σώμα της συνάρτησης.

Παράδειγμα. Ας δούμε τη συνάρτηση:

```
def f(x):
    y = 1
    x = x + y
    print('x =', x, 'y =', y)
    return x

x = 3
y = 2
z = f(x)
print('x =', x)
print('y =', y)
print('z =', z)
```

Παρατηρούμε ότι το όνομα *x* εμφανίζεται τόσο ως τυπικό όρισμα της συνάρτησης *f* αλλά και στον κώδικα ο οποίος περιέχει την κλήση της συνάρτησης (κύριο πρόγραμμα). Επίσης, το σώμα της συνάρτησης περιέχει την μεταβλητή *y* η οποία εμφανίζεται επίσης και εκτός του σώματος της συνάρτησης.

Αν εκτελέσουμε το πρόγραμμα θα δούμε τα παρακάτω αποτελέσματα

```
x = 4 y = 1
x = 3
y = 2
z = 4
```

τα οποία δείχνουν ότι οι μεταβλητές `x, y` εντός του σώματος της συνάρτησης είναι διαφορετικές από τις μεταβλητές `x, y` στο κύριο πρόγραμμα (βρίσκονται σε διαφορετικούς χώρους ονομάτων (namespace).):

Καθολικές μεταβλητές

Παράδειγμα. Ας γράψουμε μία συνάρτηση η οποία υπολογίζει τη θέση κινητού σε ελεύθερη πτώση:

```
def height(t):
    g = 9.81
    h = 0.5*g*t**2
```

```
    return h

# Main program
g = 10.0
time = float(input("Give time: "))
print(time,height(time))
```

Οι μεταβλητή *g* η οποία χρησιμοποιείται στο κύριο πρόγραμμα είναι διαφορετική από την μεταβλητή *g* η οποία χρησιμοποιείται στην συνάρτηση `height`.

- Οι υπολογισμοί στην συνάρτηση `height` γίνονται με την τιμή *g=9.81*.
- Εάν τυπώσουμε την *g* στο τέλος του κυρίου προγράμματος (μετά την κλήση της συνάρτησης) θα δούμε την τιμή *g=10.0*.

Μπορούμε να χρησιμοποιήσουμε μέσα στην συνάρτηση την τιμή *g=10.0* η οποία δίνεται στο κύριο πρόγραμμα με τον κώδικα:

```
def height(t):
    h = 0.5*g*t**2
    return h

# Main program
g = 10.0
time = float(input("Give time: "))
print(time,height(time))
```

Μπορούμε να καθορίσουμε ότι αναφερόμαστε σε μία συγκεκριμένη μεταβλητή σε διαφορετικά τμήματα του προγράμματος (π.χ., στο κύριο πρόγραμμα και σε μία συνάρτηση) δηλώνοντάς την με την εντολή `global`. Τότε το όνομα που δηλώνεται στην `global` αντιστοιχεί στην ίδια μεταβλητή στην συνάρτηση και στο κύριο πρόγραμμα.

```
def height(t):
    global g
    g = 9.81
    h = 0.5*g*t**2
    return h

# Main program
g = 10.0
```

```
time = float(input("Give time: "))
print(time,height(time))
```

Εάν τυπώσουμε την *g* στο τέλος του κυρίου προγράμματος (μετά την κλήση της συνάρτησης) θα δούμε την τιμή *g=9.81*.

Παράδειγμα. (Φυσικές σταθερές) Ας υπολογίσουμε, με τη βοήθεια συνάρτησης τη θέση σώματος σε ελεύθερη πτώση με αρχική ταχύτητα v_0. Θα οργανώσουμε το πρόγραμμα ώστε όλες οι φυσικές σταθερές να παίρνουν τιμές μέσω μίας νέας συνάρτησης.

```
# Functions
def physicalParams():
    global g,v0
    g = 10.0
    v0 = 1.0

def height(t):
    y = v0*t-0.5*g*t**2
    return y

# Main program
physicalParams()
time = float(input("Give time: "))
print(time,height(time))
```

Παράδειγμα. Θα θέλαμε να μετρήσουμε πόσες κλήσεις γίνονται στη συνάρτηση `fib` (η οποία υπολογίζει με τον αναδρομικό αλγόριθμο έναν αριθμό Fibonacci) μέσω της αναδρομικής κλήσης της.

```
def fib(n):
    global numFibCalls
    numFibCalls += 1
    if n == 0 or n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)

def testFib(n):
    global numFibCalls
    numFibCalls = 0
```

```
    print("Fibonacci number for",n,' = ',fib(n))
    print("Function fib was called",numFibCalls,"times")
```

Ερώτηση. Τρέξτε το παραπάνω πρόγραμμα (αφού προσθέσετε λίγες γραμμές που καλούν τις συναρτήσεις) και δείτε ότι ο αριθμός των κλήσεων της `fib(n)` είναι αρκετά μεγάλος. Γιατί συμβαίνει αυτό; Θα μπορούσαμε να γράψουμε διαφορετικά την `fib(n)` ώστε να χρειάζεται μικρότερος αριθμός κλήσεων;

Προδιαγραφές

Εισαγωγή. Για κάθε συνάρτηση μπορούμε να πάρουμε ένα κείμενο το οποίο αναφέρει τις *προδιαγραφές* της, δηλαδή, βασικές πληροφορίες γι' αυτήν: τον τύπο και τη σημασία των τυπικών ορισμάτων καθώς και την πληροφορία που επιστρέφει η συνάρτηση. Αυτά μπορούμε να τα ανακαλέσουμε με την εντολή `help`.

Παράδειγμα. Για τις ενσωματωμένες συναρτήσεις έχουμε:

```
>>> help(len)
Help on built-in function len in module builtins:

>>> help('math.exp')

Help on built-in function exp in math:

math.exp = exp(...)
    exp(x)

    Return e raised to the power of x.
```

Όταν γράφουμε μία συνάρτηση μπορούμε να καθορίσουμε τις προδιαγραφές της σε κείμενο το οποίο γράφουμε στις επόμενες γραμμές μετά την εντολή def και το οποίο περιέχεται ανάμεσα σε σύμβολα `"""`. (Το κείμενο αυτό λέγεται docstring.)

Παράδειγμα. Ας δούμε μία απλή συνάρτηση στην οποία θα περιλάβουμε τις προδιαγραφές της (docstring).

```
def organizeName(firstName, lastName, reverse):
    """firstName, lastName are strings, reverse is a boolean
```

```
    if reverse==False returns firstName lastName
    if reverse==True returns lastName, firstName"""

    if reverse == True:
        s = lastName + ', ' + firstName
    else:
        s = firstName + ' ' + lastName

    return s
```

Αφού τρέξουμε την συνάρτηση (ή το πρόγραμμα που περιέχει την συνάρτηση) μπορούμε να ανακαλέσουμε τις προδιαγραφές της συνάρτησης:

```
>>> help(organizeName)

organizeName(firstName, lastName, reverse)
    firstName, lastName are strings, reverse is a boolean
    if reverse==False returns firstName lastName
    if reverse==True returns lastName, firstName
```

Γραφική επανάληψη

Σχήμα. Η γενική μορφή προγράμματος με συνάρτηση και κλήση της από το κύριο πρόγραμμα.

Ορίσματα με προεπιλεγμένες τιμές (default parameter values)

Παράδειγμα. [Πηγή: Σημειώσεις Μ. Πλεξουσάκη.] Ας γράψουμε τώρα μια συνάρτηση η οποία επιστρέφει σε μια ακολουθία χαρακτήρων την ημερομηνία στη μορφή `day-month-year` ή, εναλλακτικά, στη μορφή `month-day-year`, όπως συνηθίζεται σε κάποιες χώρες του κόσμου. Είναι λογικό να θεωρήσουμε ότι η λίστα των τυπικών ορισμάτων θα περιλαμβάνει τα `day, month, year`, καθώς και ένα τέταρτο όρισμα, ας το πούμε `america` το οποίο ορίζει τον τρόπο σχηματισμού της ακολουθίας χαρακτήρων. Αν η συνάρτηση κληθεί με τιμή του ορίσματος `america` ίσο με `True` τότε θα σχηματίσει την ακολουθία χαρακτήρων `month-day-year`, διαφορετικά την `day-month-year`. (Στην Αμερική χρησιμοποιείται ο τρόπος γραφής ημερομηνίας `month-day-year`.)

```
def dateString(day, month, year, america):
    if america:
        return str(month) + '-' + str(day) + '-' + str(year)
    else:
        return str(day) + '-' + str(month) + '-' + str(year)
```

Η συνηθισμένη μορφή για την ημερομηνία (για εμάς) είναι η `day-month-year`. Η python μας δίνει τη δυνατότητα να απλοποιήσουμε την κλήση της συνάρτησης για αυτή την περίπτωση χρησιμοποιώντας προεπιλεγμένες τιμές για ένα ή περισσότερα ορίσματα (default parameter values).

```
def dateString(day, month, year, america=False):
    if america:
        return str(month) + '-' + str(day) + '-' + str(year)
    else:
        return str(day) + '-' + str(month) + '-' + str(year)
```

Η απουσία του ορίσματος `america` είναι ισοδύναμη με την κλήση της συνάρτησης με το όρισμα `america=False`. Έτσι, όλες οι παρακάτω κλήσεις της `dateString` είναι επιτρεπτές:

```
dateString(12, 10, 2016)
dateString(12, 10, 2016, True)
dateString(12, 10, 2016, america=True)
```

Η πρώτη θα επιστρέψει την ακολουθία χαρακτήρων `'12-10-2016'` ενώ οι επόμενες δύο την ακολουθία χαρακτήρων `'10-12-2016'`.

Παρατήρηση....

Keyword arguments (ορίσματα-δεσμευμένες λέξεις)

[Προαιρετικά]

Εισαγωγή. [Πηγή: Σημειώσεις Μ. Πλεξουσάκη.] Σε όλα τα παραδείγματα συναρτήσεων τα οποία έχουμε δει, η αντιστοίχιση των τυπικών ορισμάτων κατά την κλήση της

συνάρτησης γίνεται σύμφωνα με τη θέση τους (η μέθοδος αυτή αντιστοίχισης λέγεται *θεσιακή*). Η Python επιτρέπει και έναν δεύτερο τρόπο αντιστοίχισης τυπικών ορισμάτων και ορισμάτων χρησιμοποιώντας το όνομα του τυπικού ορίσματος. Θα μπορούσαμε να γράψουμε `dateString(day=12, month=10, year=2016, america=True)`. Σε αυτή την περίπτωση μπορούμε να δώσουμε τιμές στα τυπικά ορίσματα με οποιαδήποτε σειρά θέλουμε

```
dateString(year=2016, month=10, day=12, america=True)
dateString(day=12, year=2016, month=10, america=True)
dateString(america=True, day=12, year=2016, month=10)
```

Δείτε όμως ότι η κλήση `dateString(12, month=10, 2016, False)` παράγει ένα μήνυμα σφάλματος:

```
>>> dateString(12, month=10, 2016, False)
SyntaxError: positional argument follows keyword argument
```

ενώ η κλήση `dateString(12, 10, 2016, america=False)` επιστρέφει την ημερομηνία χωρίς μήνυμα σφάλματος:

```
>>> dateString(12, 10, 2016, america=False)
'12-10-2016'
```

Παρατήρηση. Εφόσον δώσουμε τιμή σε όρισμα συνάρτησης χρησιμοποιώντας το όνομά του (keyword argument) δεν μπορούμε να δώσουμε τιμή σε επόμενο όρισμα με τη μέθοδο αντιστοίχισης κατά τη θέση του ορίσματος.

Παράδειγμα. Γράψτε μία συνάρτηση στην οποία θα δίνεται όνομα και επίθετο και θα μπορεί να το τυπώνει στη μορφή *Όνομα Επίθετο* είτε *Επίθετο, Όνομα*.

```
def organizeName(firstName, lastName, reverse = False):

    if reverse == True:
        s = lastName + ', ' + firstName
    else:
```

```
            s = firstName + ' ' + lastName

        return s

    # - - - Main program - - -

    fName = input("Give your first name: ")
    lName = input("Give your last name: ")

    # call function with default argument
    s = organizeName(fName,lName)
    print(s)

    # call using keyword arguments
    s = organizeName(reverse=True, firstName='Nick',
    lastName='Papadakis')
    print(s)
```

Χρησιμοποιήσατε ένα προεπιλεγμένο όρισμα (`reverse`). Καλέσαμε τη συνάρτηση αφήνοντας την προεπιλεγμένη τιμή στο προεπιλεγμένο όρισμα. Επίσης, καλέσαμε τη συνάρτηση χρησιμοποιώντας keyword arguments με τη σειρά προτίμησής μας (και όχι με τη σειρά που εμφανίζονται τα ορίσματα στον ορισμό της συνάρτησης).

Ερώτηση. Είναι αποδεκτή η κλήση της παραπάνω συνάρτησης ως `s = organizeName(reverse=True,'Nick','Papadakis')`;

Μελέτη

Βιβλιογραφία

1. Δημήτριος Καρολίδης, *Μαθαίνετε εύκολα python* (Εκδόσεις Καρολίδη, 2016).
2. Κ. Μαγκούτης, Χ. Νικολάου, *Εισαγωγή στον αντικειμενοστραφή προγραμματισμό με Python*, (Αποθετήριο "Κάλλιπος", 2016) - Κεφάλαιο 4. Συναρτήσεις και εκτέλεση υπό συνθήκη.
3. J.V. Guttag, *Υπολογισμοί και προγραμματισμός με την python*, Κεφάλαιο 4.

Προηγούμενο Επόμενο