Κεφάλαιο 2

Μεταβλητές και Τύποι Δεδομένων

Σκοπός

Σκοπός αυτού του κεφαλαίου είναι να εισαγάγει τις θεμελιώδεις έννοιες αποθήκευσης και χειρισμού δεδομένων στην Python: μεταβλητές, τύπους δεδομένων, μορφοποίηση κειμένου και είσοδο από τον χρήστη.

Προσδοκώμενα αποτελέσματα

Μετά τη μελέτη αυτού του κεφαλαίου, θα είστε σε θέση να:

- Ορίσετε μεταβλητές και εκχωρήσετε τιμές σε αυτές.
- Αναγνωρίσετε και χρησιμοποιήσετε τους βασικούς τύπους δεδομένων (`int`, `float`, `str`, `bool`).
- Μορφοποιήσετε κείμενο με f-strings.
- Μετατρέψετε δεδομένα μεταξύ διαφορετικών τύπων.
- Διαβάσετε είσοδο από τον χρήστη με την `input()`.

Λέξεις-κλειδιά: μεταβλητές (variables), τύποι δεδομένων (data types), `int`, `float`, `str`, `bool`, f-strings, type casting, `input()`

Στο προηγούμενο κεφάλαιο γράψαμε τα πρώτα μας προγράμματα και είδαμε πώς η `print()` εμφανίζει μηνύματα στην οθόνη. Όμως τα περισσότερα προγράμματα χρειάζονται να **αποθηκεύουν δεδομένα** (αριθμούς, κείμενα, αποτελέσματα υπολογισμών) για να τα χρησιμοποιήσουν αργότερα. Αυτή ακριβώς είναι η δουλειά των **μεταβλητών**.

Συμβουλή

Πριν ξεκινήσετε, βεβαιωθείτε ότι έχετε εγκαταστήσει την Python και ότι μπορείτε να εκτελέσετε προγράμματα (Κεφάλαιο 1). Δοκιμάστε κάθε παράδειγμα στον υπολογιστή σας. Ο προγραμματισμός μαθαίνεται με πράξη!

2.1 Μεταβλητές: τι είναι και πώς τις ορίζουμε

Σκεφτείτε μια μεταβλητή σαν μια **ετικέτα** που κολλάμε πάνω σε μια τιμή. Αντί να γράφουμε κάθε φορά τον ίδιο αριθμό ή κείμενο, του δίνουμε ένα όνομα και το χρησιμοποιούμε μέσω αυτού.

2.1.1 Δημιουργία μεταβλητών

Στην Python, μια μεταβλητή δημιουργείται τη στιγμή που της αναθέτουμε μια τιμή, χρησιμοποιώντας τον τελεστή =:

```
name = "Αλίκη"
age = 20
height = 1.75

print(name)
print(age)
print(height)
```

```
Αλίκη

20
1.75
```

Συμβουλή

Μια μεταβλητή δεν είναι η ίδια η τιμή, αλλά ένα όνομα που δείχνει σε μια τιμή στη μνήμη. Έτσι, το `x = 42` καλύτερα διαβάζεται ως: «το `x` αναφέρεται στην τιμή `42`».

```
Μνήμη:     [42]    [3.14]    ["Αλίκη"]
            ^        ^           ^
         όνομα    ύψος        όνομα_φοιτητή
```

Αυτή η νοητική εικόνα θα γίνει ιδιαίτερα χρήσιμη όταν συναντήσετε μεταλλάξιμα αντικείμενα, όπου δύο ονόματα μπορούν να δείχνουν στην ίδια λίστα.

Δεν χρειάζεται να δηλώσουμε τον τύπο της μεταβλητής εκ των προτέρων, η Python τον αναγνωρίζει αυτόματα. Αυτό ονομάζεται **δυναμική τυποποίηση** (dynamic typing).

Προσοχή!

Ο τελεστής = δεν σημαίνει «ίσον» με τη μαθηματική έννοια. Σημαίνει «ανάθεσε»: πάρε την τιμή στα δεξιά και αποθήκευσέ την στη μεταβλητή στα αριστερά. Η σύγκριση ισότητας γίνεται με == (βλ. Κεφάλαιο 3).

2.1.2 Κανόνες ονομασίας μεταβλητών

Στην Python, τα ονόματα μεταβλητών πρέπει να ακολουθούν συγκεκριμένους κανόνες:

1. Μπορούν να περιέχουν **γράμματα** (a-z, A-Z), **ψηφία** (0-9) και **κάτω παύλα** (underscore _).
2. **Δεν** μπορούν να ξεκινούν με ψηφίο.
3. Είναι **case-sensitive**: η `Name` και η `name` είναι δύο διαφορετικές μεταβλητές.
4. **Δεν** μπορούμε να χρησιμοποιήσουμε δεσμευμένες λέξεις (reserved words) της Python.

```
# Έγκυρα ονόματα μεταβλητών
student_name = "Βαγγέλης"
age2 = 25
_private = "hidden"
MAX_SIZE = 100

# Μη έγκυρα ονόματα μεταβλητών (θα προκαλέσουν σφάλμα):
# 2nd_name = "error" # ξεκινά με ψηφίο
# my-name = "error" # περιέχει παύλα
# class = "error" # δεσμευμένη λέξη
```

Συμβουλή

Ακολουθήστε τη σύμβαση snake_case (μικρά γράμματα με κάτω παύλες) για μεταβλητές: `student_name`, `total_price`, `max_value`. Αυτός είναι ο προτεινόμενος τρόπος στην Python (σύμφωνα με τον PEP 8).

2.1.3 Δεσμευμένες λέξεις

Η Python έχει 35 δεσμευμένες λέξεις (keywords) που δεν μπορούμε να χρησιμοποιήσουμε ως ονόματα μεταβλητών. Μπορούμε να τις δούμε πληκτρολογώντας:

```
import keyword
print(keyword.kwlist)
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await',
'break', 'class', 'continue', 'def', 'del', 'elif', 'else',
'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in',
'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return',
'try', 'while', 'with', 'yield']
```

Δεν χρειάζεται να τις απομνημονεύσετε, ο επεξεργαστής κώδικα θα σας προειδοποιήσει αν προσπαθήσετε να τις χρησιμοποιήσετε.

2.1.4 Αλλαγή τιμής μεταβλητής

Μια μεταβλητή μπορεί να αλλάξει τιμή ανά πάσα στιγμή, ακόμα και σε διαφορετικό τύπο:

```
x = 10
print(x)

x = 20          # το x αλλάζει από 10 σε 20
print(x)

x = "hello"     # το x αλλάζει από int σε str!
print(x)
```

```
10
20
hello
```

Προσοχή!

Το ότι μια μεταβλητή *μπορεί* να αλλάξει τύπο δεν σημαίνει ότι *πρέπει*. Αυτό κάνει τον κώδικα δυσνόητο. Κατά κανόνα, κρατήστε σταθερό τον τύπο κάθε μεταβλητής.

2.1.5 Πολλαπλή ανάθεση

Η Python επιτρέπει ανάθεση σε πολλές μεταβλητές ταυτόχρονα:

```
# Πολλαπλή ανάθεση σε μία γραμμή
a, b, c = 1, 2, 3
print(a, b, c)

# Ίδια τιμή σε πολλές μεταβλητές
x = y = z = 0
print(x, y, z)

# Εναλλαγή τιμών (χωρίς βοηθητική μεταβλητή!)
a, b = 10, 20
print("Before:", a, b)

a, b = b, a
print("After:", a, b)
```

```
1 2 3
0 0 0
Before: 10 20
After: 20 10
```

Συμβουλή

Η εναλλαγή τιμών `a, b = b, a` είναι μια κομψή ιδιαιτερότητα της Python. Σε άλλες γλώσσες θα χρειαζόμασταν μια βοηθητική μεταβλητή (temp variable).

2.1.6 Η συνάρτηση type()

Για να μάθουμε τον τύπο μιας μεταβλητής, χρησιμοποιούμε τη συνάρτηση `type()`:

```
name = "Αλίκη"
age = 20
height = 1.75
is_student = True

print(type(name))
print(type(age))
print(type(height))
print(type(is_student))
```

```
<class 'str'>
<class 'int'>
<class 'float'>
<class 'bool'>
```

Στις επόμενες ενότητες θα εξετάσουμε αναλυτικά κάθε έναν από αυτούς τους τύπους δεδομένων.

2.2 Αριθμητικοί τύποι: int, float, complex

Η Python υποστηρίζει τρεις αριθμητικούς τύπους: ακέραιους, δεκαδικούς και μιγαδικούς αριθμούς.

2.2.1 Ακέραιοι αριθμοί (int)

Οι ακέραιοι αριθμοί (integers) δεν έχουν δεκαδικό μέρος:

```
x = 42
y = -17
z = 0

print(x, y, z)
print(type(x))
```

```
42 -17 0
<class 'int'>
```

Σημείωση

Σε αντίθεση με πολλές άλλες γλώσσες, οι ακέραιοι στην Python δεν έχουν όριο μεγέθους. Μπορείτε να κάνετε πράξεις με τεράστιους αριθμούς χωρίς κίνδυνο υπερχείλισης (overflow):

```
big_number = 2 ** 100
print(big_number)
```

```
1267650600228229401496703205376
```

Για μεγάλους αριθμούς, μπορούμε να χρησιμοποιήσουμε κάτω παύλες ως οπτικούς διαχωριστές, η Python τις αγνοεί:

```
population = 10_800_000
budget = 1_000_000_000
print(population)
print(budget)
```

```
10800000
1000000000
```

2.2.2 Δεκαδικοί αριθμοί (float)

Οι δεκαδικοί αριθμοί (floating-point numbers) περιέχουν δεκαδικό μέρος:

```
pi = 3.14159
temperature = -5.3
speed_of_light = 3e8      # Επιστημονική σημειογραφία: 3 * 10^8
tiny = 1.6e-19             # 1.6 * 10^-19

print(pi)
print(speed_of_light)
print(tiny)
print(type(pi))
```

```
3.14159
300000000.0
1.6e-19
<class 'float'>
```

Προσοχή!

Οι δεκαδικοί αριθμοί έχουν περιορισμένη ακρίβεια (περίπου 15-17 σημαντικά ψηφία). Αυτό μπορεί να οδηγήσει σε εκπλήξεις:

```
>>> 0.1 + 0.2
0.30000000000000004
```

Αυτό δεν είναι bug της Python, είναι θέμα αναπαράστασης δεκαδικών αριθμών στο δυαδικό σύστημα (IEEE 754 floating-point standard). Αν χρειάζεστε ακριβή δεκαδικά, δείτε τη βιβλιοθήκη `decimal`.

2.2.3 Αριθμητικές πράξεις

Ας δούμε μερικές πράξεις μεταξύ αριθμών:

```
a = 17
b = 5

print("Sum:", a + b)
print("Difference:", a - b)
print("Product:", a * b)
print("Division:", a / b)         # επιστρέφει πάντα float
```

```
print("Integer division:", a // b)
print("Remainder:", a % b)
print("Power:", a ** b)
```

```
Sum: 22
Difference: 12
Product: 85
Division: 3.4
Integer division: 3
Remainder: 2
Power: 1419857
```

Συμβουλή

Παρατηρήστε ότι η διαίρεση `/` επιστρέφει πάντα `float`, ακόμα κι αν το αποτέλεσμα είναι ακέραιος (`10 / 2` δίνει `5.0`). Για ακέραιο αποτέλεσμα χρησιμοποιήστε `//`.

2.2.4 Μιγαδικοί αριθμοί (complex)

Η Python υποστηρίζει και μιγαδικούς αριθμούς. Η φανταστική μονάδα συμβολίζεται με `j` (αντί του i που χρησιμοποιούμε στα μαθηματικά):

```
z = 3 + 4j
print(z)
print(type(z))
print("Real part:", z.real)
print("Imaginary part:", z.imag)
print("Conjugate:", z.conjugate())
print("Absolute value:", abs(z))
```

```
(3+4j)
<class 'complex'>
Real part: 3.0
Imaginary part: 4.0
Conjugate: (3-4j)
Absolute value: 5.0
```

Σημείωση

Οι μιγαδικοί αριθμοί χρησιμοποιούνται κυρίως σε μαθηματικά, φυσική και μηχανική. Αν δεν τους χρειάζεστε ακόμα, μην ανησυχείτε, θα τους ξανασυναντήσουμε στα Κεφάλαια 13 και 18.

2.3 Συμβολοσειρές (str): δημιουργία, indexing, slicing

Οι **συμβολοσειρές** (strings), γνωστές και ως αλφαριθμητικά, είναι ακολουθίες χαρακτήρων, δηλαδή κείμενο. Στην Python, ο τύπος τους είναι `str`.

2.3.1 Δημιουργία συμβολοσειρών

Μπορούμε να δημιουργήσουμε συμβολοσειρές με μονά, διπλά ή τριπλά εισαγωγικά:

```
s1 = "Hello, World!"       # διπλά εισαγωγικά
s2 = 'Hello, World!'       # μονά εισαγωγικά (ίδιο αποτέλεσμα)

# Τριπλά εισαγωγικά για string πολλών γραμμών
s3 = """This is a
multi-line
string."""

print(s1)
print(s3)
```

```
Hello, World!
This is a
multi-line
string.
```

2.3.2 Ειδικοί χαρακτήρες (escape characters)

Μερικοί χαρακτήρες δεν μπορούν να γραφτούν απευθείας μέσα σε μια συμβολοσειρά. Χρησιμοποιούμε την **ανάστροφη κάθετο** (\) ως χαρακτήρα διαφυγής (escape character):

Ακολουθία	Σημασία
\n	Νέα γραμμή (newline)
\t	Στηλοθέτης (tab)
\\	Ανάστροφη κάθετος
\'	Μονό εισαγωγικό
\"	Διπλό εισαγωγικό

```
print("First line\nSecond line")
print("Name:\Αλίκη\t")
print("Path: C:\\Users\\Documents")
print("She said: \"Hello!\"")
```

```
First line
Second line
Name:   Αλίκη
Path: C:\Users\Documents
She said: "Hello!"
```

2.3.3 Indexing: πρόσβαση σε χαρακτήρες

Κάθε χαρακτήρας σε μια συμβολοσειρά έχει μια **θέση** (index). Η αρίθμηση ξεκινάει από **0**:

P	y	t	h	o	n
0	1	2	3	4	5
−6	−5	−4	−3	−2	−1

```python
s = "Python"
print(s[0])      # πρώτος χαρακτήρας
print(s[1])      # δεύτερος χαρακτήρας
print(s[-1])     # τελευταίος χαρακτήρας
print(s[-2])     # προτελευταίος
```

```
P
y
n
o
```

Προσοχή!

Αν προσπαθήσετε να προσπελάσετε θέση εκτός ορίων, η Python θα βγάλει σφάλμα `IndexError`:

```
>>> s = "Python"
>>> s[10]
IndexError: string index out of range
```

2.3.4 Slicing: υποσυμβολοσειρές

Με το slicing μπορούμε να πάρουμε ένα **τμήμα** μιας συμβολοσειράς. Η σύνταξη είναι `s[start:stop:step]`:

```python
s = "Hello, World!"

print(s[0:5])      # χαρακτήρες 0 έως 4 (το stop εξαιρείται)
print(s[7:12])     # χαρακτήρες 7 έως 11
print(s[:5])       # από την αρχή έως 4
print(s[7:])       # από 7 έως το τέλος
print(s[::2])      # κάθε δεύτερος χαρακτήρας
print(s[::-1])     # αντεστραμμένο string!
```

```
Hello
World
Hello
World!
Hlo ol!
!dlroW ,olleH
```

Συμβουλή

Θυμηθείτε: στο `s[start:stop]`, η θέση `start` περιλαμβάνεται ενώ η θέση `stop` εξαιρείται. Ο αριθμός χαρακτήρων που παίρνουμε είναι `stop - start`.

2.3.5 Η συνάρτηση len()

Η `len()` επιστρέφει τον αριθμό χαρακτήρων σε μια συμβολοσειρά:

```
s = "Python"
print(len(s))

empty = ""
print(len(empty))

with_spaces = "Hello World"
print(len(with_spaces))    # τα κενά μετρούν!
```

```
6
0
11
```

2.3.6 Οι συμβολοσειρές είναι αμετάβλητες (immutable)

Δεν μπορούμε να αλλάξουμε μεμονωμένο χαρακτήρα σε μια συμβολοσειρά:

```
s = "Hello"
# s[0] = "h" # TypeError: το αντικείμενο 'str' δεν
                  # υποστηρίζει εκχώρηση στοιχείου
```

Αν θέλουμε μια τροποποιημένη εκδοχή, δημιουργούμε νέα συμβολοσειρά:

```
s = "Hello"
s_new = "h" + s[1:]
print(s_new)
```

```
hello
```

2.3.7 Συνένωση και επανάληψη

Ο τελεστής `+` ενώνει δύο συμβολοσειρές (concatenation) και ο `*` επαναλαμβάνει μια συμβολοσειρά:

```
first = "Hello"
second = "World"
greeting = first + " " + second
print(greeting)

line = "-" * 30
print(line)
```

```
Hello World
------------------------------
```

2.4 Βασικές μέθοδοι συμβολοσειρών

Οι συμβολοσειρές στην Python διαθέτουν δεκάδες ενσωματωμένες **μεθόδους** (methods), συναρτήσεις που καλούνται πάνω στην ίδια τη συμβολοσειρά με τη σύνταξη `s.method()`.

2.4.1 Αλλαγή πεζών/κεφαλαίων

```
s = "hello, world"

print(s.upper())       # όλα κεφαλαία
print(s.lower())       # όλα πεζά
print(s.title())       # κεφαλαίο σε κάθε λέξη
print(s.capitalize())  # κεφαλαίο μόνο τον πρώτο χαρακτήρα
print(s.swapcase())    # εναλλαγή πεζών <-> κεφαλαίων
```

```
HELLO, WORLD
hello, world
Hello, World
Hello, world
HELLO, WORLD
```

Προσοχή!

Οι μέθοδοι αυτές δεν αλλάζουν την αρχική συμβολοσειρά (αφού οι συμβολοσειρές είναι αμετάβλητες). Επιστρέφουν μια νέα συμβολοσειρά. Αν θέλετε να κρατήσετε το αποτέλεσμα, αναθέστε το σε μεταβλητή:

```
s = s.upper()   # overwrite s with the uppercase version
```

2.4.2 Αφαίρεση κενών

```
s = "   Hello, World!   "

print(repr(s.strip()))    # αφαίρεση κενών από τα δύο άκρα
print(repr(s.lstrip()))   # αφαίρεση κενών μόνο αριστερά
print(repr(s.rstrip()))   # αφαίρεση κενών μόνο δεξιά
```

```
'Hello, World!'
'Hello, World!   '
'   Hello, World!'
```

Συμβουλή

Η `repr()` εμφανίζει τη συμβολοσειρά με τα εισαγωγικά της, ώστε να φαίνονται τα κενά στην αρχή και στο τέλος. Πολύ χρήσιμη για debugging!

2.4.3 split() και join()

Η `split()` σπάει μια συμβολοσειρά σε λίστα «λέξεων» και η `join()` τα ξανασυνδέει:

```
sentence = "Python is a great language"
```

```
words = sentence.split()     # διαχωρισμός με κενά
print(words)

# Διαχωρισμός με συγκεκριμένο χαρακτήρα
data = "Αλίκη,25,Athens"
parts = data.split(",")
print(parts)

# Ένωση λίστας σε συμβολοσειρά
result = " - ".join(words)
print(result)
```

```
['Python', 'is', 'a', 'great', 'language']Αλίκη
['', '25', 'Athens']
Python - is - a - great - language
```

2.4.4 Αναζήτηση και αντικατάσταση

```
s = "Hello, World! Hello, Python!"

print(s.find("World"))      # θέση πρώτης εμφάνισης
print(s.find("Java"))       # -1 αν δεν βρεθεί
print(s.count("Hello"))     # πλήθος εμφανίσεων
print(s.replace("Hello", "Hi"))
```

```
7
-1
2
Hi, World! Hi, Python!
```

2.4.5 Έλεγχοι αρχής/τέλους

```
filename = "report.pdf"

print(filename.startswith("report"))
print(filename.endswith(".pdf"))
print(filename.endswith(".txt"))
```

```
True
True
False
```

2.4.6 Έλεγχοι χαρακτήρων

```
print("123".isdigit())        # μόνο ψηφία;
print("abc".isalpha())        # μόνο γράμματα;
print("abc123".isalnum())     # γράμματα ή/και ψηφία;
print("HELLO".isupper())      # όλα κεφαλαία;
print("hello".islower())      # όλα πεζά;
print("   ".isspace())        # μόνο κενά;
```

```
True
```

```
True
True
True
True
True
```

2.4.7 Αλυσιδωτή κλήση μεθόδων (method chaining)

Μπορούμε να καλέσουμε πολλές μεθόδους στη σειρά:

```
raw = "   Hello, World!   "
result = raw.strip().lower().replace("world", "python")
print(result)
```

```
hello, python!
```

Αυτό λειτουργεί γιατί κάθε μέθοδος επιστρέφει μια νέα συμβολοσειρά, πάνω στην οποία καλείται η επόμενη μέθοδος.

Σημείωση

Η Python διαθέτει πολλές ακόμα μεθόδους συμβολοσειρών. Μπορείτε να δείτε την πλήρη λίστα πληκτρολογώντας `dir(str)` στην κονσόλα ή επισκεπτόμενοι την τεκμηρίωση: https://docs.python.org/3/library/stdtypes.html#string-methods.

2.5 Μορφοποίηση συμβολοσειρών: f-strings, .format()

Πολύ συχνά θέλουμε να ενσωματώσουμε τιμές μεταβλητών μέσα σε κείμενο. Η Python προσφέρει πολλούς τρόπους, αλλά ο καλύτερος είναι τα **f-strings**.

2.5.1 f-strings (ο προτιμώμενος τρόπος)

Ένα f-string ξεκινάει με `f` πριν τα εισαγωγικά. Μέσα σε αγκύλες `{}` βάζουμε εκφράσεις που αποτιμώνται αυτόματα:

```
name = "Αλίκη"
age = 20
print(f"My name is {name} and I am {age} years old.")
```

```
My name is Αλίκη and I am 20 years old.
```

Μέσα στις αγκύλες μπορούμε να βάλουμε **οποιαδήποτε έκφραση**, όχι μόνο μεταβλητές:

```
x = 10
y = 3
print(f"{x} + {y} = {x + y}")
print(f"{x} * {y} = {x * y}")

name = "αλίκη"
print(f"Hello, {name.title()}!")
```

```
10 + 3 = 13
10 * 3 = 30
Hello, Αλίκη!
```

2.5.2 Μορφοποίηση αριθμών

Τα f-strings επιτρέπουν λεπτομερή έλεγχο στη μορφοποίηση αριθμών, χρησιμοποιώντας **προσδιοριστές μορφοποίησης** (format specifiers) μετά το `:`:

```
pi = 3.14159265

# Δεκαδικά ψηφία
print(f"pi = {pi:.2f}")       # 2 δεκαδικά ψηφία
print(f"pi = {pi:.4f}")       # 4 δεκαδικά ψηφία

# Ποσοστό
score = 0.875
print(f"Score: {score:.1%}")   # ως ποσοστό

# Διαχωριστικό χιλιάδων
population = 10800000
print(f"Population: {population:,}")

# Μηδενικά αριστερά
order = 42
print(f"Order #{order:05d}")   # 5 ψηφία, συμπλήρωση με μηδενικά
```

```
pi = 3.14
pi = 3.1416
Score: 87.5%
Population: 10,800,000
Order #00042
```

2.5.3 Στοίχιση κειμένου

Μπορούμε να στοιχίσουμε κείμενο σε πεδίο συγκεκριμένου πλάτους:

```
name = "Αλίκη"
score = 95.5

# Δεξιά στοίχιση σε πεδίο 10 χαρακτήρων
print(f"|{name:>10}|")

# Αριστερά στοίχιση
print(f"|{name:<10}|")

# Κεντράρισμα
print(f"|{name:^10}|")

# Γέμισμα με χαρακτήρα
print(f"|{name:*^10}|")

# Μορφοποίηση σε πίνακα
print(f"{'Name':<10} {'Score':>6}")
print(f"{'-' * 17}")
```

```
print(f"Αλίκη{'':<10} {95.5:>6.1f}")
print(f"Βαγγέλης{'':<10} {87.3:>6.1f}")
print(f"Γιώργος{'':<10} {92.0:>6.1f}")
```

```
|     Αλίκη|Αλίκη
|     |
|  Αλίκη   |Αλίκη
|*****|
Name        Score
----------------Αλίκη
        95.5Βαγγέλης
     87.3Γιώργος
      92.0
```

Η σύνταξη της στοίχισης ακολουθεί τη μορφή «γέμισμα, σύμβολο, πλάτος»: το `>` ευθυγραμμίζει δεξιά, το `<` αριστερά, το `^` κεντράρει. Ο αριθμός που ακολουθεί ορίζει το πλάτος της στήλης σε χαρακτήρες. Όταν το περιεχόμενο υπερβαίνει το πλάτος, η Python δεν το περικόπτει — το πλάτος λειτουργεί ως ελάχιστο, όχι ως μέγιστο.

Συμβουλή

Τα f-strings είναι ο πιο ευανάγνωστος και αποδοτικός τρόπος μορφοποίησης στην Python. Χρησιμοποιήστε τα παντού!

2.5.4 Η μέθοδος .format()

Πριν τα f-strings (Python 3.6+), ο κύριος τρόπος μορφοποίησης ήταν η μέθοδος `.format()`:

```
name = "Βαγγέλης"
age = 25

# Ορίσματα θέσης
print("My name is {} and I am {} years old.".format(name, age))

# Αριθμημένα ορίσματα
print("{0} is {1} years old. Hello, {0}!".format(name, age))

# Οι προσδιοριστές μορφοποίησης λειτουργούν όμοια
pi = 3.14159
print("pi = {:.2f}".format(pi))
```

```
My name is Βαγγέλης and I am 25 years old.Βαγγέλης
 is 25 years old. Hello, Βαγγέλης!
pi = 3.14
```