Python Programming

Hellenic Mediterranean University

Lecture 2

Dr. Alina Eqtami

Lecture Roadmap

Σήμερα:

- Arithmetic operators και αριθμητική συμπεριφορά
- Order of operations (αυστηρά)
- Mixing types και type casting
- Boolean λογική και συγκρίσεις
- if ως έλεγχος ροής
- for και range σε βάθος
- Reverse iteration
- Accumulation patterns
- while ως state evolution

Στόχος: Να μπορείτε να προβλέπετε εκτέλεση χωρίς Run.

Arithmetic Operators

```
print(7 + 3)
print(7 - 3)
print(7 * 3)
print(7 / 3)
print(7 // 3)
print(7 % 3)
print(2 ** 5)
```

Ερώτηση: Τι διαφορά έχει / και //;

Negative Floor Division

```
print(-7 // 3)
print(-7 % 3)
```

Σημαντικό: Floor division = round down προς -∞
Όχι truncate.

// and % as Engineering Tools – Χρήση σε χρόνο

```
seconds = 3671

hours = seconds // 3600
rem = seconds % 3600

minutes = rem // 60
sec = rem % 60

print(hours, minutes, sec)
```

Trace:
3671 = 1 ώρα, 1 λεπτό, 11 δευτερόλεπτα

// → ακέραιο μέρος
% → υπόλοιπο (remainder)

Order of Operations

Python follows:

$$() \rightarrow ** \rightarrow */// \rightarrow +-$$

Παράδειγμα:

$$2 + 3 * 4 = 14$$

$$(2 + 3) * 4 = 20$$

Right Associativity

```
print(2 ** 3 ** 2)
```

Είναι:

$$2^{(3^2)} = 2^9 = 512$$

Όχι:

$$(2^3)^2$$

Numeric Types

- int → ακριβής ακέραιος
- float → αριθμητική προσέγγιση

Η μεταβλητή δεν έχει τύπο. Η τιμή έχει τύπο.

Mixing Types

```
print(10 + 2.5)
print(10 / 2)
print(10 // 2.0)
```

Κανόνας: Αν υπάρχει float → αποτέλεσμα float

Type Conversion

```
x = 3.7

print(int(x))
print(round(x))
print(float(5))
```

int → truncate

round → nearest

Float Trap (Precision) – Παγίδα float

```
print(0.1 + 0.2)
print(0.1 + 0.2 == 0.3)
```

Σημείο-κλειδί: τα float είναι προσέγγιση (binary representation).
Άρα μια ”φαινομενικά ίση” πράξη μπορεί να μην είναι ακριβώς ίση.

Σήμερα: απλώς το παρατηρούμε. Αργότερα θα κάνουμε tolerance ($|a-b|<\epsilon$).

Comparison Operators

== != > < >= <=

Επιστρέφουν: True ή False

Boolean Evaluation

```
print(5 > 3)
print(5 == 3)
print(3 <= 3)
print(4 != 4)
```

Logical Operators

and or not

Παράδειγμα:

$$(x > 0) \wedge (x < 10)$$

Logical Trace

```
x = 7

print(x > 5 and x < 10)
print(x < 5 or x == 7)
print(not(x == 7))
```

Trace mentally.

if Statement

```
if condition:
   block
else:
   block
```

Μόνο ένα branch εκτελείται.

Parity Example

```
x = 12

if x % 2 == 0:
   print("Even")
else:
   print("Odd")
```

Προβλέψτε.

Multiple Branches

```
grade = 85

if grade >= 90:
   print("A")
elif grade >= 80:
   print("B")
else:
   print("C")
```

Transition to Loops

Επανάληψη = επαναλαμβανόμενη αλλαγή state.
Θέλουμε: Να εκτελούμε block πολλές φορές.

for Loop Structure

```
for i in range(...):
    block
```

range → παράγει ακολουθία.

range(stop)

range(10)
Παράγει:
0,1,2,3,4,5,6,7,8,9
10 στοιχεία.

Why 0 to N-1?

Αν θέλω N επαναλήψεις:
range(N)
Παράγει N αριθμούς.
Μηχανική συνέπεια: Αρίθμηση από 0.

range(N) – Τι παράγει πραγματικά;

```
N = 5

for i in range(N):
   print(i)
```

Trace:

Iteration	i
1	0
2	1
3	2
4	3
5	4

Παράγει ακριβώς N αριθμούς.
Ξεκινά από 0 και σταματά πριν το N.

Counting 10 Objects

```
for i in range(10):
    print(i)
```

10 αντικείμενα → 0 έως 9

0→N-1 vs 1→N – Ποια η διαφορά;

```
N = 5

print("range(N):")
for i in range(N):
    print(i)

print("range(1, N+1):")
for i in range(1, N+1):
    print(i)
```

Παρατήρηση:
range(N) → 0,1,2,3,4 (N στοιχεία)
range(1,N+1) → 1,2,3,4,5 (N στοιχεία)

Το ένα μετράει θέσεις (indexing).
Το άλλο μετράει φυσικούς αριθμούς.

range(start, stop)

range(1, 11)
Παράγει:
1 έως 10
stop δεν περιλαμβάνεται.

1 to N

```
N = 10

for i in range(1, N+1):
    print(i)
```

Off-by-One Error

range(10) → 0..9 range(1,10) → 1..9 range(1,11) → 1..10
Πάντα ρωτάμε: Περιλαμβάνεται το stop;

range with Step

range(start, stop, step)
Παράδειγμα:
range(0,20,5)
0,5,10,15

Reverse Iteration

```
for i in range(10, 0, -1):
    print(i)
```

10,9,8,...,1

Reverse range – Τι παράγει πραγματικά;

```
for i in range(5, 0, -1):
    print(i)
```

Trace:

Iteration	i
1	5
2	4
3	3
4	2
5	1

Κανόνας:
Όταν step < 0, πρέπει start > stop.
Το stop (0) δεν περιλαμβάνεται.

Reverse Off-by-One – Συχνό λάθος

```
print("range(5, 0, -1):")
for i in range(5, 0, -1):
    print(i)

print("range(5, -1, -1):")
for i in range(5, -1, -1):
    print(i)
```

Παρατήρηση:
range(5, 0, -1) → 5,4,3,2,1
range(5, -1, -1) → 5,4,3,2,1,0

Το stop δεν περιλαμβάνεται.
Άρα για να φτάσω στο 0, βάζω stop = -1.

Reverse Rule

Αν step < 0:
start > stop
range(10,0,-1)

Summation Pattern

```
total = 0

for i in range(1,6):
    total = total + i

print(total)
```

Pattern: accumulation

Summation: 0→N-1 vs 1→N

```
N = 5

sum1 = 0
for i in range(N):
    sum1 = sum1 + i

sum2 = 0
for i in range(1, N+1):
    sum2 = sum2 + i

print(sum1, sum2)
```

Υπολογιστικά:
range(N) → 0+1+2+3+4 = 10
range(1,N+1) → 1+2+3+4+5 = 15

Προσοχή: Το μαθηματικό $\sum_{i=1}^{N} i \sum_{i=0}^{N-1} i$.

Reverse Summation

```
total = 0

for i in range(5,0,-1):
   total = total + i

print(total)
```

Το αποτέλεσμα είναι ίδιο. Η ροή διαφέρει.

General Summation

$$\sum_{i=1}^{N} i$$

Κώδικας:
range(1, N+1)

while Loop

```
while condition:
    block
```

Επανάληψη βασισμένη σε condition.

Exponential Growth

```
x = 1

while x < 100:
    x = x * 2

print(x)
```

Infinite Loop Example

```
x = 5

while x > 0:
    print(x)
```

Δεν αλλάζει state. Δεν τελειώνει.

for vs while

for: Γνωστό πλήθος επαναλήψεων
while: Άγνωστο πλήθος Βασίζεται σε condition

Engineering Example

Υπολογισμός factorial:

$$N! = 1 \cdot 2 \cdot 3 \cdots N$$

Πώς θα το γράφατε;

Factorial with for

```
N = 5
fact = 1

for i in range(1, N+1):
    fact = fact * i

print(fact)
```

Factorial: Forward vs Reverse

```
N = 5

# Forward
fact1 = 1
for i in range(1, N+1):
    fact1 = fact1 * i

# Reverse
fact2 = 1
for i in range(N, 0, -1):
    fact2 = fact2 * i

print(fact1, fact2)
```

Παρατήρηση:
Το αποτέλεσμα είναι ίδιο (120).

Αλλά η σειρά state evolution διαφέρει.

Takeaways

- Order matters
- Types matter
- range excludes stop
- 0→N-1 φυσική αρίθμηση
- Loops change state