Τελευταία ενημέρωση 12/10/2025

Python – Μέρος Α
ΕΚΔΔΑ

1.3 Εισαγωγή στην Python

Εβδομάδα 1/7

Σεπτέμβριος 2025

Προγραμματισμός και γλώσσες προγραμματισμού

- Προγραμματισμός είναι η διαδικασία δημιουργίας προγραμμάτων
- Πρόγραμμα είναι η περιγραφή μιας σειράς βημάτων που επιτελούν έναν σκοπό (συνήθως παράγουν αποτελέσματα υπολογισμών)
- Τα προγράμματα γράφονται σε γλώσσες προγραμματισμού (π.χ., Python, Java, C++, C, Rust κ.α.)

Μεταγλώττιση και διερμηνεία προγραμμάτων

- Ο πηγαίος κώδικας δεν μπορεί να εκτελεστεί απευθείας σε ένα υπολογιστικό σύστημα, πρέπει να μεταφραστεί σε γλώσσα μηχανής
- Υπάρχουν 2 βασικές κατηγορίες μεταφραστικών προγραμμάτων:
  - **μεταγλωττιστές (compilers)**: μετάφραση όλου του πηγαίου κώδικα σε γλώσσα μηχανής, εκτέλεση του παραγόμενου εκτελέσιμου κώδικα
  - **διερμηνευτές (interpreters)**: μετάφραση και εκτέλεση εντολή προς εντολή του πηγαίου κώδικα
- Οι σχεδιαστές κάθε γλώσσας προγραμματισμού επιλέγουν σε ποιο βαθμό η γλώσσα βρίσκεται πλησιέστερα στη μια ή στην άλλη κατηγορία (μεταγλωττιζόμενη ή διερμηνευόμενη)

Η γλώσσα προγραμματισμού Python

- Οι εκδόσεις της Python 2 & 3
  - Η έκδοση 2 και η έκδοση 3 της Python δεν είναι συμβατές μεταξύ τους
  - Η τελευταία έκδοση της Python 2 ήταν η Python 2.7 (έφτασε σε EOL=End Of Life στις 1/1/2020), συνεπώς δεν δέχεται βελτιώσεις και διορθώσεις (π.χ. για θέματα ασφάλειας!)
- Πλέον δεν χρησιμοποιείται η Python 2, ωστόσο υπάρχουν πολλές εφαρμογές, βιβλιοθήκες και κώδικας που έχουν γραφεί σε Python 2 και δεν έχουν μεταφερθεί σε Python 3
- Στη συγκεκριμένη επιμόρφωση θα χρησιμοποιήσουμε Python 3

Παραδείγματα διαφορετικής συμπεριφοράς σε Python 2 και Python 3

Χαρακτηριστικό	Python 2	Python 3
Εκτύπωση	print "Hello" (η print είναι εντολή)	print("Hello") (η print είναι συνάρτηση)
Διαίρεση (/)	5/2 → 2 (ακέραια διαίρεση)	5/2 → 2.5 (πραγματική διαίρεση)
	5.0/2 → 2.5	5//2 → 2 (ακέραια διαίρεση)
Συμβολοσειρές	"abc" → str (bytes)	"abc" → str (Unicode)
	u"abc" → unicode	b"abc" → bytes
Είσοδος	raw_input() → συμβολοσειρά	input() → συμβολοσειρά
	input() → αποτιμά κώδικα	raw_input() αφαιρέθηκε στην έκδοση 3
Range	range(5) → λίστα	range(5) → lazy (σαν το xrange)
	xrange(5) → οκνηρή αποτίμηση	xrange αφαιρέθηκε στην έκδοση 3
Εξαιρέσεις	except ValueError, e:	except ValueError as e:
Μέθοδοι λεξικών	d.keys() → λίστα	d.keys() → dict_keys (view)
	d.items() → λίστα	d.items() → dict_items (view)
Unicode	Δεν είναι το default (πρέπει να χρησιμοποιηθεί το u"...")	Default (όλα τα λεκτικά είναι Unicode)
Διάσχιση	map(), zip(), filter() → λίστες	map(), zip(), filter() → iterators

Υλοποιήσεις της Python

- **CPython: πρόκειται για την υλοποίηση αναφοράς και default υλοποίηση της Python - https://www.python.org/**
  - **Έχει γραφεί σε C**
  - **Μεταγλωττίζει αρχεία .py σε .pyc και στη συνέχεια τα εκτελεί (διερμηνευόμενα, εντολή προς εντολή) στην εικονική μηχανή CPython**
  - **Παρουσιάζει υψηλή σταθερότητα και έχει διαθεσιμότητα μεγάλου αριθμού βιβλιοθηκών**
  - **Έχει σχετικά χαμηλή ταχύτητα εκτέλεσης, κυρίως λόγω διερμηνείας και GIL=Global Interpreter Lock**
- **PyPy**: πρόκειται για τη 2η πιο σημαντική υλοποίηση της Python - https://pypy.org/
  - Παράγει, σε ορισμένες περιπτώσεις, σημαντικά ταχύτερο κώδικα από ότι η CPython
  - Παρουσιάζει θέματα συμβατότητας με καθιερωμένες βιβλιοθήκες της Python (π.χ. numpy, pandas, scikit-learn κ.α.)
- **Cython**: πρόκειται για υπερσύνολο της γλώσσας Python με προαιρετικές δηλώσεις τύπων
  - Χρησιμοποιείται για την επιτάχυνση αριθμητικών υπολογισμών
  - Απαιτεί ένα επιπλέον βήμα μεταγλώττισης και είναι λιγότερο δυναμική σε σχέση με την Python
- Άλλες υλοποιήσεις: **Jython, IronPython, MicroPython, CircuitPython, RustPython, Nuitka** κ.α.

REPL (1/2) – Τι είναι;

- Εφόσον έχει εγκατασταθεί η Python (βλ. Διαφάνειες 10,11 από Παρουσίαση 1.2), μπορεί να χρησιμοποιηθεί το REPL
- Ενεργοποιείται απλά γράφοντας python στη γραμμή εντολών
- REPL σημαίνει:
  - **R**ead – λήψη εισόδου από τον χρήστη (python κώδικας)
  - **E**val – αποτίμηση κώδικα
  - **P**rint – εκτύπωση αποτελεσμάτων
  - **L**oop – επανάληψη της διαδικασίας μέχρι να γίνει έξοδος από τον χρήστη
- Χρήσιμο για:
  - γρήγορο πειραματισμό με κώδικα
  - αποσφαλμάτωση κώδικα
  - δοκιμή συναρτήσεων βιβλιοθηκών
  - κλήση συστήματος βοήθειας, π.χ. >>> help(str)

REPL (2/2) – Παραδείγματα κώδικα Python

- Άμεση εκτέλεση κώδικα
- Χωρίς ανάγκη να γραφεί πλήρες πρόγραμμα (script)
- Πλοήγηση στο ιστορικό των εντολών με τα βελάκια πάνω και κάτω
- Επιτρέπεται η εισαγωγή εντολών πολλαπλών γραμμών
- Έξοδος με exit() ή με Ctrl+D
- Από την Python 3.13 το REPL διαθέτει syntax highlighting

```
>>> 2 + 3**2
11
>>> "hello".upper()
'HELLO'
```

Παραδείγματα στο REPL με modules της Python Standard Library

- math
- random
- datetime
- calendar

```
>>> import random
>>> random.uniform(0,1)
0.20503277568409528
>>> random.uniform(0,1)
0.9468137833893536
>>> random.randint(1,6)
4
>>> random.randint(1,6)
1
>>> random.choice([1,2,3,4,5,6])
3
>>> random.choice([1,2,3,4,5,6])
1
```

```
>>> import calendar
>>> cal = calendar.month(2025, 9)
>>> print(cal)
   September 2025
Mo Tu We Th Fr Sa Su
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30
```

```
>>> import math
>>> math.sqrt(2)
1.4142135623730951
>>> math.pi
3.141592653589793
>>> math.e
2.718281828459045
>>> math.factorial(5)
120
>>> math.factorial(50)
30414093201713378043612608166064768844377641568960512000000000000
>>> math.comb(6,2)
15
```

```
>>> import datetime
>>> d = datetime.date(1990, 5, 17)
>>> d
datetime.date(1990, 5, 17)
>>> dt = datetime.datetime(1990, 5, 17, 14, 30)
>>> dt
datetime.datetime(1990, 5, 17, 14, 30)
>>> delta = datetime.timedelta(days=10, hours=5)
>>> dt + delta
datetime.datetime(1990, 5, 27, 19, 30)
>>> dt.year
1990
>>> dt.month
5
>>> dt.day
17
```

Άσκηση #1: Εκφώνηση

- Υπολογίστε τις ακόλουθες εκφράσεις στο REPL και παρατηρήστε τα αποτελέσματα που λαμβάνετε:

```
>>> 2 + 3 * 4
>>> "Python"[::-1]
>>> [x**2 for x in range(6)]
>>> import random; random.randint(1,6)
>>> random.choice(["πέτρα", "ψαλίδι", "χαρτί"])
>>> sorted([7, 2, 3, 1, 6])
```

- Δοκιμάστε στο REPL τις ακόλουθες 4 εντολές μια προς μια:

```
>>> import this
>>> import __hello__; __hello__.main()
>>> from __future__ import braces
>>> import antigravity
```

Άσκηση #1: Αποτελέσματα εντολών στο REPL

```
$ python
Python 3.13.7 | packaged by conda-forge | (main, Sep  3 2
025, 14:24:46) [Clang 19.1.7 ] on darwin
Type "help", "copyright", "credits" or "license" for more
 information.
>>> 2 + 3 * 4
14
>>> "Python"[::-1]
'nohtyP'
>>> [x**2 for x in range(6)]
[0, 1, 4, 9, 16, 25]
>>> import random; random.randint(1,6)
4
>>> import random; random.randint(1,6)
3
>>> random.choice(["πέτρα", "ψαλίδι", "χαρτί"])
'χαρτί'
>>> random.choice(["πέτρα", "ψαλίδι", "χαρτί"])
'χαρτί'
>>> sorted([7, 2, 3, 1, 6])
[1, 2, 3, 6, 7]
```

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the
```

```
>>> import __hello__
>>> __hello__.main()
Hello world!
>>> from __future__ import braces
  File "<python-input-10>", line 1
    from __future__ import braces
                           ^^^^^^
SyntaxError: not a chance
>>> import antigravity
```

https://xkcd.com/353/

Μεταβλητές (1/2)

- Στην Python οι μεταβλητές (variables) είναι ονοματισμένες αναφορές προς αντικείμενα που βρίσκονται στη μνήμη του υπολογιστή
- Λόγω του dynamic typing, δεν δηλώνονται τύποι δεδομένων, ενώ ο τύπος των μεταβλητών μπορεί να αλλάζει καθώς εκτελείται ο κώδικας
- Η ανάθεση τιμών σε μεταβλητές γίνεται με τον τελεστή =, ο δε τύπος της μεταβλητής προκύπτει από τον τύπο του αντικειμένου στο δεξί μέρος της ανάθεσης
- Οι μεταβλητές έχουν όνομα (name), τύπο (type) και αναγνωριστικό (id):
  - Με τη χρήση της συνάρτησης type() βλέπουμε τον τύπο μιας μεταβλητής
  - Με τη χρήση της συνάρτησης id() βλέπουμε το αναγνωριστικό «θέσης» μνήμης μιας μεταβλητής

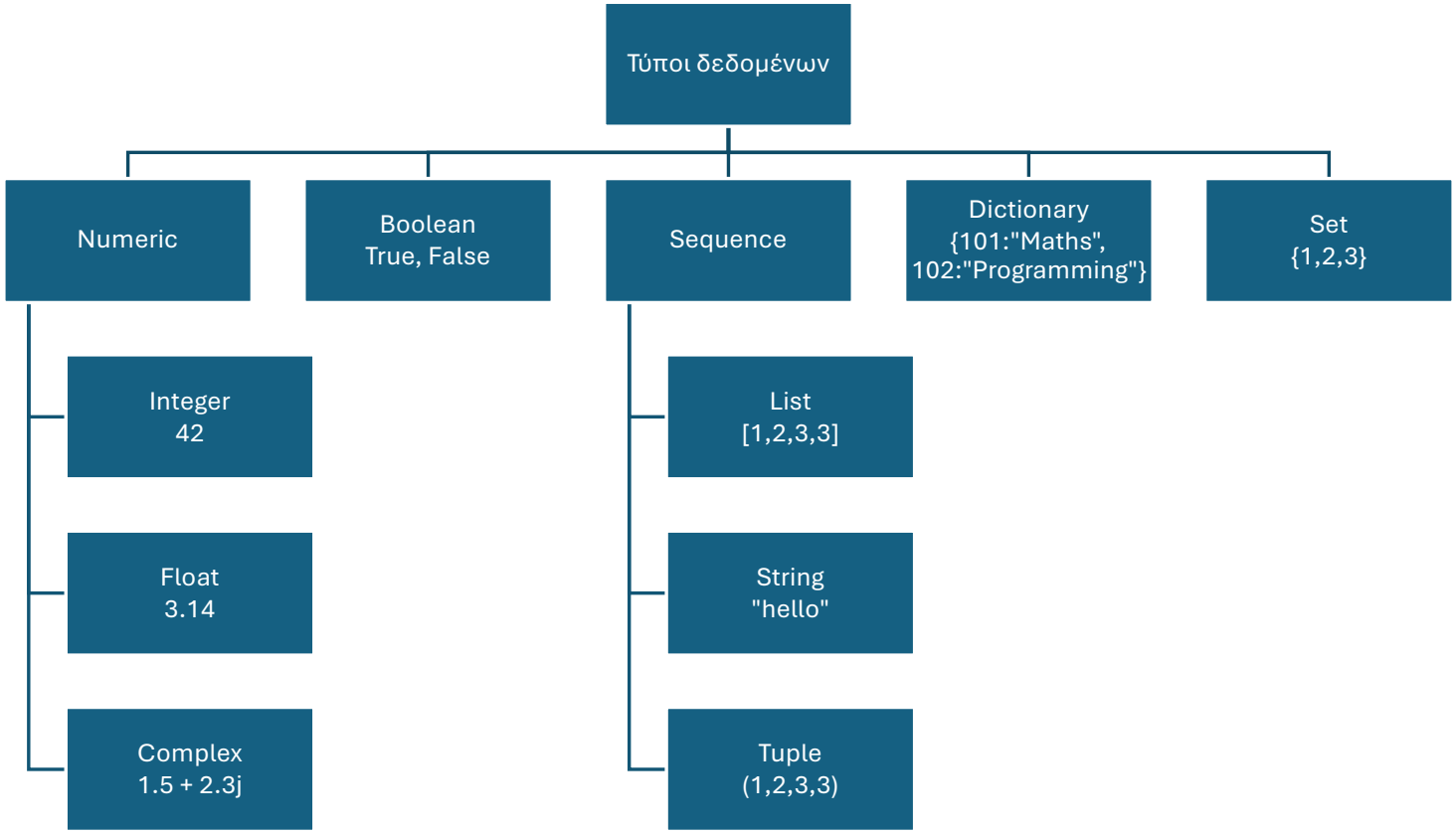
```
>>> x = 42
>>> type(x)
<class 'int'>
>>> id(x)
4306464400
>>> x = "python"
>>> type(x)
<class 'str'>
>>> id(x)
4310246208
```

Μεταβλητές (2/2)

- Τα ονόματα των μεταβλητών ξεκινούν με γράμμα ή με κάτω παύλα _ (underscore) και μπορούν να περιέχουν γράμματα ψηφία και την κάτω παύλα
- Στις μεταβλητές, αλλά και γενικότερα, υπάρχει διάκριση πεζών κεφαλαίων
- Μπορούν να χρησιμοποιηθούν ελληνικοί χαρακτήρες στα ονόματα των μεταβλητών, αλλά αυτό δεν συνίσταται

```
>>> a_valid_variable_name = 42
>>> a_valid_variable_name
42
>>> ένα_έγκυρο_όνομα_μεταβλητής = 7
>>> ένα_έγκυρο_όνομα_μεταβλητής
7
>>> x = 1729
>>> X
Traceback (most recent call last):
  File "<python-input-37>", line 1, in <module>
    X
NameError: name 'X' is not defined. Did you mean: 'x'?
```

Τύποι δεδομένων της Python



Αριθμητικοί τελεστές

Σύμβολο	Πράξη	Παράδειγμα στο REPL
+	Πρόσθεση	>>> 1 + 2 3
-	Αφαίρεση	>>> 1 – 2 -1
*	Πολλαπλασιασμός	>>> 2 * 3 6
/	Διαίρεση	>>> 10 / 4 2.5
//	Πηλίκο ακέραιας διαίρεσης	>>> 10 // 3 3
%	Υπόλοιπο ακέραιας διαίρεσης	>>> 10 % 3 1
**	Ύψωση σε δύναμη	>>> 2 ** 10 1024

Τετραγωνική ρίζα, κυβική ρίζα κ.ο.κ.

- Για να υπολογιστεί η τετραγωνική ρίζα ενός αριθμού γνωρίζουμε ότι ο αριθμός θα πρέπει να υψωθεί στη δύναμη 1/2
- Ομοίως για την κυβική ρίζα υψώνεται στη δύναμη 1/3 κ.ο.κ. για ρίζες υψηλότερης τάξης

```
>>> 2**0.5
1.4142135623730951
>>> 2**(1/2)
1.4142135623730951
>>> 2**(1/3)
1.2599210498948732
>>> 2**(1/4)
1.189207115002721
>>> 2**(1/5)
1.148698354997035
>>> -2**0.5
-1.4142135623730951
>>> (-2)**0.5
(8.659560562354934e-17+1.4142135623730951j)
>>> import math
>>> math.sqrt(2)
1.4142135623730951
>>> math.sqrt(-2)
Traceback (most recent call last):
  File "<python-input-82>", line 1, in <module>
    math.sqrt(-2)
    ~~~~~~~~~^^^^
ValueError: math domain error
>>> import cmath
>>> cmath.sqrt(-2)
1.4142135623730951j
```

Προτεραιότητα αριθμητικών τελεστών

- Ισχύει ότι ισχύει στα μαθηματικά για τις αριθμητικές πράξεις, δηλαδή, πρώτα παρενθέσεις, μετά ύψωση σε δύναμη, μετά πολλαπλασιασμός και διαίρεση και τέλος πρόσθεση και αφαίρεση
- Σε πράξεις με τελεστές ίδιας προτεραιότητας, στην πρόσθεση/αφαίρεση, καθώς και στο πολλαπλασιασμό/διαίρεση οι πράξεις γίνονται από αριστερά προς τα δεξιά, ενώ στην ύψωση σε δύναμη οι πράξεις γίνονται από δεξιά προς τα αριστερά

```
>>> 10/4/2
1.25
>>> (10/4)/2
1.25
>>> 10/(4/2)
5.0
>>> 2**1**2
2
>>> 2**(1**2)
2
>>> (2**1)**2
4
```

Αυτόματη μετατροπή τύπων

- Ο τύπος δεδομένων μιας μεταβλητής μπορεί να αλλάζει αυτόματα εφόσον αυτό υποδεικνύεται από τις πράξεις που γίνονται
- Στο παράδειγμα η μεταβλητή x σταδιακά αποκτά τύπο δεδομένων:
  ακέραιο → πραγματικό → μιγαδικό

```
>>> x = 2
>>> type(x)
<class 'int'>
>>> x = x / 3
>>> x
0.6666666666666666
>>> type(x)
<class 'float'>
>>> x = -x
>>> x = x**0.5
>>> x
(4.9995996217394874e-17+0.816496580927726j)
>>> type(x)
<class 'complex'>
```

Εξαναγκασμένη αλλαγή τύπου μεταβλητής

- Υπάρχουν περιπτώσεις που μπορεί να είναι επιθυμητή η αλλαγή τύπου μιας μεταβλητής (π.χ. από ακέραιο σε χαρακτήρες, από πραγματικό σε ακέραιο κ.α.)
- Αυτό ονομάζεται type casting και γίνεται όπως φαίνεται στα παραδείγματα δεξιά

```
>>> x = 10
>>> type(x)
<class 'int'>
>>> x = str(x)
>>> type(x)
<class 'str'>
>>> y = 10/3
>>> type(y)
<class 'float'>
>>> y = int(y)
>>> type(y)
<class 'int'>
```

Είσοδος τιμών από τον χρήστη (1/2)

- Με την ενσωματωμένη συνάρτηση input() ζητείται από τον χρήστη να εισάγει δεδομένα από το πληκτρολόγιο
- Τα δεδομένα επιστρέφονται από την input() ως λεκτικά, οπότε αν χρειάζεται να χρησιμοποιηθούν στη συνέχεια ως ακέραιες τιμές ή πραγματικές τιμές θα πρέπει να γίνει η κατάλληλη μετατροπή τύπων

```
>>> x = input()
7
>>> x, type(x)
('7', <class 'str'>)
>>> x = int(input())
7
>>> x, type(x)
(7, <class 'int'>)
```

Είσοδος τιμών από τον χρήστη (2/2)

- Η input() μπορεί να δέχεται ως όρισμα ένα λεκτικό που χρησιμοποιείται ως μήνυμα προς τον χρήστη
- Με αυτό τον τρόπο μπορεί να γραφεί κώδικας που καθοδηγεί τον χρήστη για τις τιμές που πρέπει να εισάγει

```
>>> x = float(input("Δώσε μια πραγματική τιμή: "))
Δώσε μια πραγματική τιμή: 3.14159
>>> x
3.14159
>>> type(x)
<class 'float'>
```

Εμφάνιση αποτελεσμάτων με την print (1/2)

- Η ενσωματωμένη συνάρτηση print() χρησιμοποιείται για εμφάνιση τιμών και μηνυμάτων στην οθόνη σε μια γραμμή
- Η print() μπορεί να δέχεται πολλαπλά ορίσματα που εμφανίζονται το ένα δίπλα στο άλλο με ένα κενό χαρακτήρα ως διαχωριστικό
- Μπορούν να χρησιμοποιηθούν οι παράμετροι sep και end για να τροποποιήσουν τη συμπεριφορά της print()

```
>>> a, b = 1, 2
>>> print(a, "message", a + b)
1 message 3
>>> print(a, "message", a + b, sep="-")
1-message-3
>>> print(a, "message", a + b, end="")
1 message 3>>>
```

Εμφάνιση αποτελεσμάτων με την print (2/2)

- Είναι συνηθισμένο να προετοιμάζεται μια συμβολοσειρά εξόδου με κατάλληλη μορφοποίηση της ώστε να περιέχει τιμές μεταβλητών και εκφράσεων σε προκαθορισμένες θέσεις πριν εμφανιστεί στην οθόνη με την print()
- Η μορφοποίηση μπορεί να γίνει:
  - Με το σύμβολο %
  - Με τη μέθοδο των συμβολοσειρών .format()
  - Με f-strings (νεότερος και προτιμότερος τρόπος), δείτε το https://docs.python.org/3/tutorial/inputoutput.html

```
>>> name, age = "Μαρία", 31
>>> s1 = "Το όνομα είναι %s και η ηλικία είναι %s" % (name, age)
>>> s1
'Το όνομα είναι Μαρία και η ηλικία είναι 31'
>>> s2 = "Το όνομα είναι {} και η ηλικία είναι {}".format(name, age)
>>> s2
'Το όνομα είναι Μαρία και η ηλικία είναι 31'
>>> s3 = f"Το όνομα είναι {name} και η ηλικία είναι {age}"
>>> s3
'Το όνομα είναι Μαρία και η ηλικία είναι 31'
```

Σχόλια

- Τα σχόλια (comments) στον κώδικα, αγνοούνται από τον διερμηνευτή και χρησιμοποιούνται για να παρέχουν εξήγηση σε σημεία του κώδικα που ο προγραμματιστής κρίνει ότι απαιτείται
- Η έναρξη σχολίου στην Python γίνεται με το σύμβολο #
- Μια καλή πρακτική είναι το σχόλιο να μην εξηγεί το τι κάνει ο κώδικας αλλά γιατί το κάνει
- Επίσης, τα σχόλια θα πρέπει να είναι καθαρά διατυπωμένα, ενημερωμένα και σύντομα

Συμβολοσειρές

- Μια ακολουθία χαρακτήρων που οριοθετείται (αρχή και τέλος) με ειδικά σύμβολα ονομάζεται συμβολοσειρά
- Στην Python για την αρχή και το τέλος της συμβολοσειράς μπορούν να χρησιμοποιηθούν, μονά εισαγωγικά (π.χ. 'test'), διπλά εισαγωγικά (π.χ. "test"), τρία μονά εισαγωγικά (π.χ. '''test'''), και τρία διπλά εισαγωγικά (π.χ. """test""")
- Τα τριπλά εισαγωγικά χρησιμοποιούνται για συμβολοσειρές που καταλαμβάνουν πολλές γραμμές
- Μπορεί να χρησιμοποιηθεί συνδυασμός από μονά και διπλά εισαγωγικά για να υπάρχουν μέσα στη συμβολοσειρά είτε μονά είτε διπλά εισαγωγικά
- Οι συμβολοσειρές συχνά αναφέρονται ως αλφαριθμητικά ή λεκτικά

```
>>> s1 = 'this is a test'
>>> print(s1)
this is a test
>>> s2 = "this is a test"
>>> print(s2)
this is a test
>>> s1 == s2
True
>>> s3 = '''this is
... a test'''
>>> print(s3)
this is
a test
>>> s4 = """this is
... a test"""
>>> print(s4)
this is
a test
>>> s5 = 'this is a "test"'
>>> print(s5)
this is a "test"
```

Δεικτοδότηση συμβολοσειρών

- Οι χαρακτήρες που περιέχονται σε μια συμβολοσειρά έχουν δείκτες με τους οποίους μπορούν να αναφερθούν
- Η δεικτοδότηση (indexing) ξεκινά για τον πρώτο χαρακτήρα από τα αριστερά από το 0 και αυξάνεται κατά ένα για κάθε χαρακτήρα προς τα δεξιά
- Υπάρχει και εναλλακτική δεικτοδότηση από το τέλος προς την αρχή με αρνητικούς δείκτες, ξεκινά με τον πλέον δεξιό χαρακτήρα που έχει δείκτη -1 και ο δείκτης μειώνεται κατά ένα για κάθε χαρακτήρα προς τα αριστερά

'this is a test’

t	h	i	s		i	s		a		t	e	s	t
0	1	2	3	4	5	6	7	8	9	10	11	12	13
-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
>>> s = 'this is a test'
>>> s[0], s[5], s[8], s[13]
('t', 'i', 'a', 't')
>>> s[-1], s[-14]
('t', 't')
```

Τεμαχισμός συμβολοσειρών

- Ο τεμαχισμός (slicing) συμβολοσειρών επιστρέφει ένα τμήμα μιας συμβολοσειράς με βάση το μοτίβο [<αρχή>:<τέλος>:<βήμα>]
  - Η <αρχή>, <τέλος>, <βήμα> πρέπει να είναι ακέραιοι
  - Αν παραλείπεται η <αρχή> υπονοείται η αρχή της συμβολοσειράς
  - Αν παραλείπεται το <τέλος> υπονοείται το τέλος της συμβολοσειράς
  - Αν παραλείπεται το <βήμα> υπονοείται η τιμή 1
- Προσοχή! το τμήμα s[start:end], ξεκινά από το start αλλά δεν περιέχει το χαρακτήρα στη θέση end, σταματά στο end -1, δηλαδή είναι συμπεριληπτικό (inclusive) του start, αλλά όχι του end (exclusive)

s = 'this is a test’

t	**h**	**i**	**s**		**i**	**s**		**a**		**t**	**e**	**s**	**t**
0	1	2	3	4	5	6	7	8	9	10	11	12	13
-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Τεμαχισμός	**Ερμηνεία**	**παράδειγμα**
s[start:end:step]	Τμήμα της s από τον δείκτη start μέχρι και τον δείκτη end-1	s[5:7] → 'is'
s[start:]	Τμήμα της s από τον δείκτη start μέχρι το τέλος	s[8:] → 'a test'
s[:end]	Τμήμα της s από την αρχή μέχρι μέχρι και τον δείκτη end-1	s[:4] → 'this'
s[:]	Αντιγραφή της συμβολοσειράς	s[:] → 'this is a test'
s[::-1]	Λήψη αντίστροφής συμβολοσειράς	s[::-1] → 'tset a si siht’

Η συναρτήση len() και διάφορες μέθοδοι συμβολοσειρών

- Η συνάρτηση len() επιστρέφει το μήκος μιας συμβολοσειράς
- Μέθοδοι συμβολοσειρών (καλούνται σε αντικείμενα συμβολοσειρών, χρησιμοποιώντας την τελεία και το όνομα της μεθόδου)
  - .upper()
  - .lower()
  - .find(str, start, end)
  - .count(str, start, end)
  - .index(value)
  - .replace(old, new, count)
- Δείτε για λεπτομέρειες το https://docs.python.org/3/library/stdtypes.html#string-methods

t	h	i	s		i	s		a		t	e	s	t
0	1	2	3	4	5	6	7	8	9	10	11	12	13
-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
>>> s = 'this is a test'
>>> len(s)
14
>>> s.upper()
'THIS IS A TEST'
>>> s.find('is')
2
>>> s.find('is',3)
5
>>> s.find('is',8)
-1
>>> s.count('t')
3
>>> s.index('e')
11
```

```
>>> s = 'this is a test'
>>> s.replace("t", "T", 2)
'This is a Test'
>>> s.replace(" ", "")
'thisisatest'
```

Συγκριτικοί τελεστές

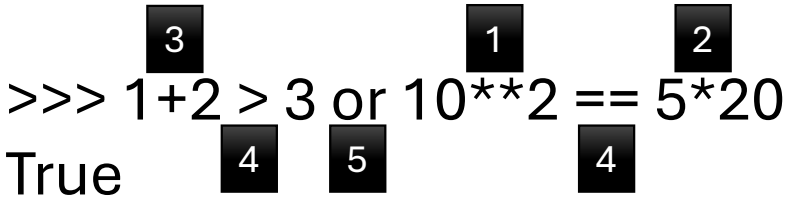
Συγκριτικός τελεστής	Ερμηνεία
a == b	Έλεγχος αν οι τιμές των μεταβλητών a και b είναι ίσες
a != b	Έλεγχος αν οι τιμές των μεταβλητών a και b είναι διαφορετικές
a < b	Έλεγχος αν η τιμή της a είναι μικρότερη από την τιμή της b
a > b	Έλεγχος αν η τιμή της a είναι μεγαλύτερη από την τιμή της b
a <= b	Έλεγχος αν η τιμή της a είναι μικρότερη ή ίση από την τιμή της b
a >= b	Έλεγχος αν η τιμή της a είναι μεγαλύτερη ή ίση από την τιμή της b
a is b	Έλεγχος αν η a και η b αναφέρονται στο ίδιο αντικείμενο
a in b	Έλεγχος αν η a υπάρχει στη b (π.χ. 'a' in 'banana' επιστρέφει True)

Λογικοί τελεστές (boolean)

Λογικός τελεστής	Ερμηνεία	Πίνακας αληθείας
and	λογικό ΚΑΙ (σύζευξη)	False and False → False False and True → False True and False → False True and True → True
or	λογικό Ή (διάζευξη)	False or False → False False or True → True True or False → True True or True → True
not	λογική άρνηση	not False → True not True → False

Προτεραιότητα ανάμεσα σε αριθμητικούς, λογικούς και συγκριτικούς τελεστές

- Η προτεραιότητα σε εκφράσεις με διάφορα είδη τελεστών είναι:
  1. Αριθμητικοί τελεστές (υψηλότερη)
  2. Συγκριτικοί τελεστές
  3. Λογικοί τελεστές (χαμηλότερη)



Τελεστές συμβολοσειρών

- Συνένωση συμβολοσειρών (+)
- Επανάληψη (*)
- Σχέση μέλους (in, not in)
- Σύγκριση συμβολοσειρών (λεξικογραφικά με τους συγκριτικούς τελεστές)
- Δεικτοδότηση και τεμαχισμός ([], [:])
- Επαυξημένη ανάθεση (+=, *=)

```
>>> 'Hello' + ' World!'
'Hello World!'
>>> 'hi ' * 5
'hi hi hi hi hi '
>>> 'day' in 'birthday'
True
>>> 'apple' < 'appricot'
True
>>> s = 'Hello'
>>> s += ' World!'
>>> s
'Hello World!'
```

Τελεστές που εφαρμόζονται σε δυαδικά ψηφία

- Δυαδικό ΚΑΙ (&) – πρέπει και οι 2 δυαδικά ψηφία να είναι 1 για να είναι το αποτέλεσμα 1, αλλιώς το αποτέλεσμα είναι 0
- Δυαδικό Ή (|) – αρκεί ένα από τα 2 δυαδικά ψηφία να είναι 1 για να είναι το αποτέλεσμα 1, αλλιώς το αποτέλεσμα είναι 0
- Αποκλειστικό Ή (^) – πρέπει τα 2 δυαδικά ψηφία να είναι διαφορετικά μεταξύ τους για να είναι το αποτέλεσμα 1, αλλιώς είναι 0
- Αριστερή ολίσθηση (<<) – η ολίσθηση κατά 1 ψηφίο προς τα αριστερά προκαλεί διπλασιασμό του αριθμού
- Δεξιά ολίσθηση (>>) – η ολίσθηση κατά 1 ψηφίο προς τα δεξιά προκαλεί ακέραιο υποδιπλασιασμό του αριθμού

```
>>> x, y = 17, 5
>>> f'{x:05b}' # εμφάνιση του 17 στη δυαδική του μορφή
'10001'
>>> f'{y:05b}' # εμφάνιση του 5 στη δυαδική του μορφή με 5 ψηφία
'00101'
>>> f'{x & y:05b}' # δυαδικό ΚΑΙ
'00001'
>>> f'{x | y:05b}' # δυαδικό Ή
'10101'
>>> f'{x ^ y:05b}' # δυαδικό XOR
'10100'
>>> f'{x << 1} {x << 1:b}' # αριστερή ολίσθηση του 17 για 1 δυαδικό ψηφίο
'34 100010'
>>> f'{x >> 1} {x >> 1:b}' # δεξιά ολίσθηση του 17 για 1 δυαδικό ψηφίο
'8 1000'
```

Η εντολή επιλογής if

- Η εντολή επιλογής if επιτρέπει την εκτέλεση ενός μπλοκ εντολών με βάση την αποτίμηση μιας συνθήκης
- Μπλοκ εντολών είναι μια ή περισσότερες εντολές που βρίσκονται στο ίδιο επίπεδο κατακόρυφης στοίχισης (indentation) και που εκτελούνται μαζί
- Μια εντολή if μπορεί να βρίσκεται μέσα σε ένα μπλοκ μιας άλλη εντολής if (εμφωλευμένα if)

```
if <συνθήκη1>:
        <μπλοκ εντολών 1>
elif <συνθήκη2>:
        <μπλοκ εντολών 2>
elif <συνθήκη3>:
        <μπλοκ εντολών 3>
...
else:
        <μπλοκ εντολών n>
```

Παράδειγμα με την εντολή if

- Πρόγραμμα που δέχεται μια τιμή από τον χρήστη και εμφανίζει το πρόσημο της τιμής

```
x = input("Εισάγετε έναν αριθμό: ")
x = int(x)
if x > 0:
    print("Θετικός")
elif x < 0:
    print("Αρνητικός")
else:
    print("Μηδέν")
```

```
Εισάγετε έναν αριθμό: 57
Θετικός
```

Ένα ακόμα παράδειγμα με την εντολή if

- Πρόγραμμα που δέχεται δύο βαθμούς από τον χρήστη και υπολογίζει το μέσο όρο τους. Αν οι βαθμοί απέχουν πάνω από 2 μονάδες τότε ο μικρότερος αλλάζει έτσι ώστε να απέχει 2 βαθμούς από τον μεγαλύτερο βαθμό (π.χ. αν οι βαθμοί είναι 16 και 17 τότε ο μέσος όρος είναι 16.5, αν οι βαθμοί είναι 10 και 20, τότε το 10 γίνεται 18 και ο μέσος όρος (18+20)/2 = 19)

```
a = float(input("Δώσε τον πρώτο βαθμό: "))
b = float(input("Δώσε το δεύτερο βαθμό: "))
if a > b:
    a, b = b, a  # στο a βρίσκεται η
    # μικρότερη τιμή και
    # στο b η μεγαλύτερη τιμή

if b - a > 2:
    a = b - 2
    c = (a + b) / 2
    print(f"Ο προσαρμοσμένος Μ.Ο. είναι {c}")
else:
    c = (a + b) / 2
    print(f"Ο Μ.Ο. είναι {c}")
```

```
Δώσε τον πρώτο βαθμό: 10
Δώσε το δεύτερο βαθμό: 20
Ο προσαρμοσμένος μέσος όρος είναι 19.0
```

Διπλές ανισότητες σε συνθήκες

- Η Python υποστηρίζει έναν συντομότερο τρόπο γραφής διπλών ανισοτήτων, όπως χρησιμοποιούνται και στα μαθηματικά
- Για παράδειγμα η συνθήκη που ελέγχει ότι η θερμοκρασία είναι μεγαλύτερη από 18 αλλά μικρότερη ή ίση του 26 μπορεί να γραφεί ως **t > 18 and t <= 26** ή ως **18 < t <= 26**
- Συνήθως χρησιμοποιείται ο δεύτερος τρόπος (διπλή ανισότητα), για συντομία

Ο τριαδικός τελεστής στην Python

- Ο τριαδικός τελεστής (ternary operator ή conditional expression) είναι ένα σύντομος τρόπος να γραφεί μια εντολή if/else σε μια γραμμή
- Η σύνταξή του είναι η ακόλουθη:

  τιμή_αν_αληθής if συνθήκη else τιμή_αν_ψευδής
- Πρόκειται για έναν συνοπτικό τρόπο που καθιστά τον κώδικα ευανάγνωστο για **απλές συνθήκες**, για αποτύπωση συνθετότερης λογικής να αποφεύγεται

- Πρόγραμμα που εμφανίζει μήνυμα «Ενήλικος» ή «Ανήλικος» ανάλογα με τη ηλικία

```
age = 20
status = "Ενήλικος" if age >= 18 else "Ανήλικος"
print(status)  # Ενήλικος
```

Άσκηση #2: Εκφώνηση

- Γράψτε πρόγραμμα που να δέχεται από το χρήστη έναν ακέραιο αριθμό και να εμφανίζει με κατάλληλο μήνυμα σε ποια από τις ακόλουθες περιπτώσεις βρίσκεται ο αριθμός:
  - Άρτιος και θετικός
  - Άρτιος και αρνητικός
  - Περιττός και θετικός
  - Περιττός και αρνητικός
  - Μηδέν

Άσκηση #2: Λύση

```
num = int(input("Δώσε έναν ακέραιο αριθμό: "))

if num == 0:
    print("Ο αριθμός είναι μηδέν")
elif num > 0 and num % 2 == 0:
    print("Ο αριθμός είναι άρτιος και θετικός")
elif num < 0 and num % 2 == 0:
    print("Ο αριθμός είναι άρτιος και αρνητικός")
elif num > 0 and num % 2 != 0:
    print("Ο αριθμός είναι περιττός και θετικός")
else:
    print("Ο αριθμός είναι περιττός και αρνητικός")
```

```
Δώσε έναν ακέραιο αριθμό: 1789
Ο αριθμός είναι περιττός και θετικός
```

Η εντολή επανάληψης while

- Η εντολή while επιτρέπει την επαναληπτική εκτέλεση ενός μπλοκ κώδικα (το <μπλοκ εντολών 1>) για όσο ισχύει μια συνθήκη, όταν η συνθήκη γίνει ψευδής, τότε η εκτέλεση συνεχίζεται με τις εντολές μετά την while
- Στην Python έχει προστεθεί στην while και το προαιρετικό τμήμα else:
- Αν υπάρχει σε μια εντολή while το else: και το <μπλοκ εντολών 2>, τότε αυτό θα εκτελεστεί αν η συνθήκη γίνει ψευδής και μόνο τότε (δηλαδή αν δεν γίνει έξοδος από την επανάληψη νωρίτερα με την εντολή break που θα δούμε στη συνέχεια)

```
while <συνθήκη>:
        <μπλοκ εντολών 1>
else:
        <μπλοκ εντολών 2>
```

Παράδειγμα με την εντολή while

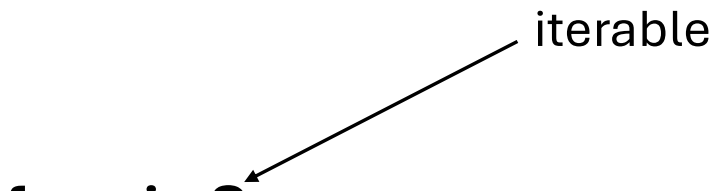
- Πρόγραμμα που δέχεται μια ακέραια τιμή και πραγματοποιεί αντίστροφη μέτρηση μέχρι το 0. Στο τέλος να εμφανίζει το μήνυμα «Τέλος εκτέλεσης»

```
x = int(input("Δώσε μια θετική ακέραια τιμή: "))
while x >= 0:
    print(f"{x}")
    x -= 1  # ισοδύναμο με x = x - 1
print("Τέλος εκτέλεσης")
```

```
Δώσε μια θετική ακέραια τιμή: 10
10
9
8
7
6
5
4
3
2
1
0
Τέλος εκτέλεσης
```

Η εντολή επανάληψης for

- Η εντολή for επιτρέπει την επαναληπτική εκτέλεση του ίδιου μπλοκ κώδικα για κάθε στοιχείο μιας ακολουθίας (ενός iterable όπως για παράδειγμα μια συμβολοσειρά, μια λίστα κ.α.)
- Γενικά, η for χρησιμοποιείται όταν το πλήθος των επαναλήψεων είναι γνωστό εκ των προτέρων
- Το τμήμα else της εντολής λειτουργεί όπως και στη while, δηλαδή το <μπλοκ εντολών 2> εκτελείται όταν η έξοδος από την επανάληψη προκληθεί λόγω εξάντλησης των στοιχείων της ακολουθίας και όχι αν προκληθεί πρόωρη έξοδος από την επανάληψη (με την break)




```
for x in S
        <μπλοκ εντολών 1>
else:
        <μπλοκ εντολών 2>
```

Παράδειγμα με την εντολή for

- Πρόγραμμα που εμφανίζει το κάθε ψηφίο του αριθμού 1234567890 σε νέα γραμμή, μετά την εμφάνιση όλων των ψηφίων εμφανίζει το κείμενο «Τέλος»

```
a = 1234567890
for x in str(a):
    print(x)
print("Τέλος")
```

```
1
2
3
4
5
6
7
8
9
0
Τέλος
```

Οι εντολές break και continue

- Η εντολή **break** διακόπτει την εκτέλεση του βρόχου επανάληψης στον οποίο βρίσκεται και προκαλεί τη συνέχεια εκτέλεσης με την πρώτη εντολή μετά το βρόχο
- Η εντολή **continue** διακόπτει την εκτέλεση της τρέχουσας επανάληψης του βρόχου επανάληψης στον οποίο βρίσκεται και προκαλεί τη συνέχεια εκτέλεσης της επόμενης επανάληψης του βρόχου

```
s = '12345'
print("Break")
for c in s:
        if c == '3':
                break
        print(c)


print("Continue")
for c in s:
        if c == '3':
                continue
        print(c)
```

```
Break
1
2
Continue
1
2
4
5
```

Η συνάρτηση range()

- Η συνάρτηση range(start, end, step) επιτρέπει τη δημιουργία ακολουθιών ακεραίων αριθμών
  - Το όρισμα start συμπεριλαμβάνεται στην ακολουθία, αν παραληφθεί υπονοείται το 0
  - Το όρισμα end δεν μπορεί να παραληφθεί, και η τιμή end δεν συμπεριλαμβάνεται στην ακολουθία
  - Το όρισμα step καθορίζει το βήμα της ακολουθίας (start, start + step, start + 2* step, ...), αν παραληφθεί υπονοείται το 1

```
>>> range(10)
range(0, 10)
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(1,10,3)) # από το 1 μέχρι το 9 με βήμα 3
[1, 4, 7]
>>> list(range(10,0,-1)) # από το 10 μέχρι το 1 με βήμα -1
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Εμφωλευμένες επαναλήψεις

- Προπαίδεια από 1 έως 5 (μια εκτύπωση με for και μια με while)

```
for i in range(1, 6):
        for j in range(1, 6):
                print(f"{i * j:2}", end=" ")
        print()

print("-" * 20)

i = 1
while i <= 5:
        j = 1
        while j <= 5:
                print(f"{i * j:2}", end=" ")
                j += 1
        print()
        i += 1
```

```
 1  2  3  4  5
 2  4  6  8 10
 3  6  9 12 15
 4  8 12 16 20
 5 10 15 20 25
--------------------
 1  2  3  4  5
 2  4  6  8 10
 3  6  9 12 15
 4  8 12 16 20
 5 10 15 20 25
```

Άσκηση #3: Εκφώνηση

- Γράψτε κώδικα που να επιλύει το πρόβλημα εντοπισμού ενός τυχαίου ακέραιου αριθμού από το 1 μέχρι το 100, με 6 το πολύ προσπάθειες του χρήστη. Για κάθε επιλογή του χρήστη θα λαμβάνει ανατροφοδότηση σχετικά με το αν η επιλογή του πέτυχε τη ζητούμενη τιμή ή αν είναι μικρότερη ή μεγαλύτερη. Αν ο χρήστης εντοπίσει τη ζητούμενη τιμή σε κάποια από τις 6 προσπάθειες θα ανακηρύσσεται νικητής, αλλιώς θα έχει χάσει το παιχνίδι. Για τη δημιουργία ενός τυχαίου αριθμού (lucky_number) στο διάστημα [1,100] χρησιμοποιήστε τον ακόλουθο κώδικα:

```
import random
lucky_number = random.randint(1,100)
```

Άσκηση #3: Λύση

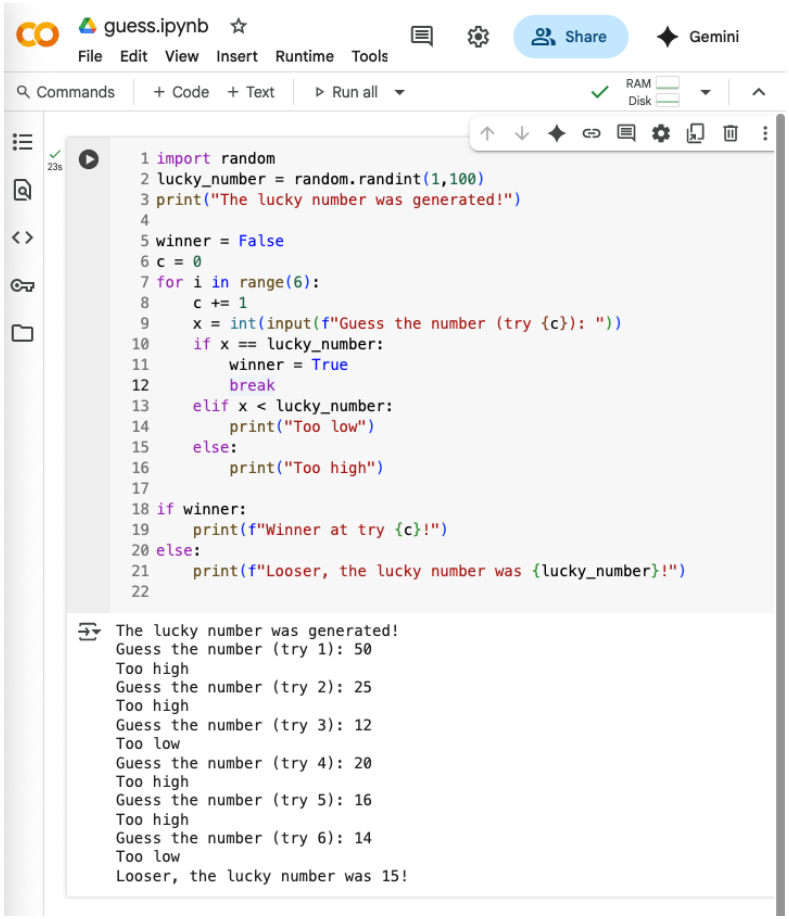
```python
import random
lucky_number = random.randint(1,100)
print("The lucky number was generated!")

winner = False
c = 0
for i in range(6):
    c += 1
    x = int(input(f"Guess the number (try {c}): "))
    if x == lucky_number:
        winner = True
        break
    elif x < lucky_number:
        print("Too low")
    else:
        print("Too high")

if winner:
    print(f"Winner at try {c}!")
else:
    print(f"Looser, the lucky number was {lucky_number}!")
```

```
The lucky number was generated!
Guess the number (try 1): 50
Too low
Guess the number (try 2): 75
Too high
Guess the number (try 3): 62
Too low
Guess the number (try 4): 69
Too low
Guess the number (try 5): 73
Too low
Guess the number (try 6): 74
Winner at try 6!
```

Παράδειγμα επίλυσης της άσκησης 3 στο Google Colab



https://colab.research.google.com/drive/1uy1dx0-AwRF1g3Uh1XVOe7uTecYwkcVF?usp=sharing

Παράδειγμα επίλυσης της άσκησης 3 στο VS Code