Ενότητα 2

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΜΕ ΤΗ ΓΛΩΣΣΑ PYTHON

Το περιβάλλον EduBlocks

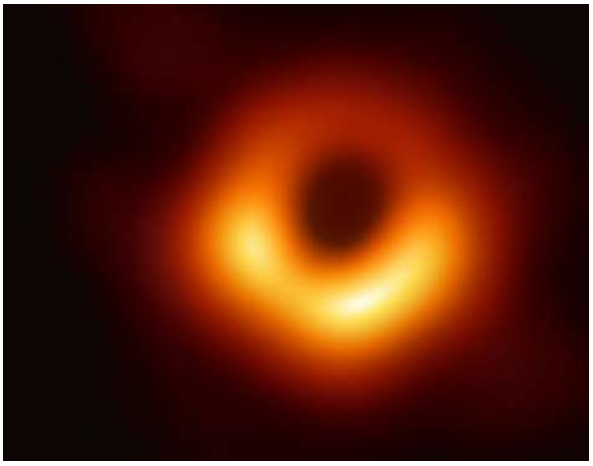
Η γλώσσα Python

Ανάπτυξη εφαρμογών

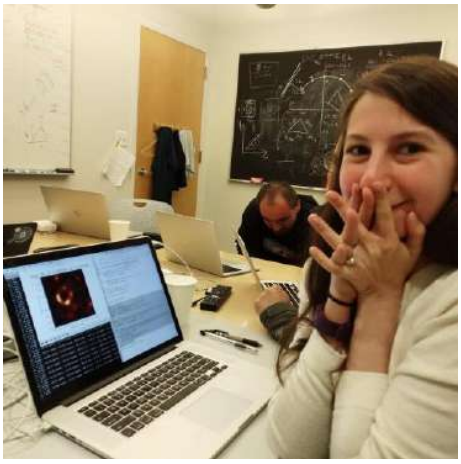
Ενότητα 2. Προγραμματισμός με τη γλώσσα Python

2.1 Εισαγωγή

Καθημερινά χρησιμοποιούμε διάφορες εφαρμογές, οι οποίες μας διευκολύνουν σε πολλές εργασίες. Αν θέλουμε να βρούμε πληροφορίες για κάτι που δε γνωρίζουμε χρησιμοποιούμε μια μηχανή αναζήτησης, όπως η Google ή ένα γλωσσικό μοντέλο Τεχνητής Νοημοσύνης, όπως το ChatGPT. Αν θέλουμε να αποθηκεύσουμε τα αρχεία μας σε έναν δικτυακό φάκελο στο υπολογιστικό νέφος, ώστε να έχουμε πρόσβαση από παντού στα αρχεία μας, χρησιμοποιούμε μια εφαρμογή διαμοιρασμού αρχείων, όπως είναι το DropBox. Για ψυχαγωγία χρησιμοποιούμε εφαρμογές, όπως το Spotify για μουσική ή το Instagram για επικοινωνία ή το Netflix, για να δούμε ταινίες ή το youtube, για να δούμε ή να ανεβάσουμε βίντεο. Όλες αυτές οι εφαρμογές έχουν κάτι κοινό. Είναι γραμμένες στην ίδια γλώσσα προγραμματισμού, τη γλώσσα Python. Η Python είναι μια υψηλού επιπέδου γλώσσα προγραμματισμού, η οποία δημιουργήθηκε από τον Ολλανδό Guido van Rossum το 1990. Η γλώσσα Python χρησιμοποιείται και σε επιστημονικές εφαρμογές, όπως για παράδειγμα στην αστροφυσική, όπου οι επιστήμονες οπτικοποίησαν για πρώτη φορά μια μαύρη τρύπα.



Εικόνα 2.1. Η πρώτη εικόνα μαύρης τρύπας (M87) που δημιουργήθηκε από λογισμικό Τεχνητής Νοημοσύνης γραμμένο στη γλώσσα Python από την ερευνητική ομάδα του Event Horizon Telescope



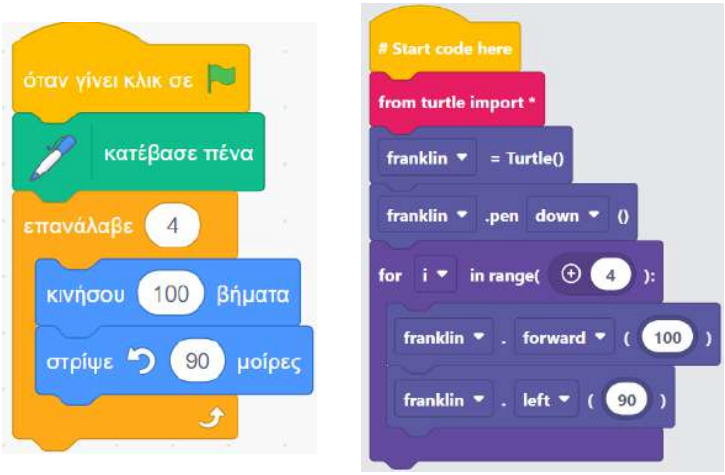
Εικόνα 2.2. Η ερευνήτρια Katie Bouman μόλις βλέπει για πρώτη φορά το αποτέλεσμα του αλγορίθμου οπτικοποίησης της μαύρης τρύπας που ανέπτυξε η ίδια στη γλώσσα Python

Σε αυτήν την ενότητα θα γράψετε τα πρώτα σας προγράμματα σε μια πραγματική γλώσσα προγραμματισμού η οποία χρησιμοποιείται ευρέως από προγραμματιστές, μηχανικούς και επιστήμονες για την ανάπτυξη χρήσιμων εφαρμογών.

2.2 Η γλώσσα Python

Η Python είναι μια υψηλού επιπέδου γλώσσα προγραμματισμού η οποία δημιουργήθηκε από τον Ολλανδό Guido van Rossum το 1990. Ένα σημαντικό πλεονέκτημά της είναι η ευκολία χρήσης της. Για τον λόγο αυτό χρησιμοποιείται από πολλά πανεπιστήμια και σχολεία σε εισαγωγικά μαθήματα αλγοριθμικής.

Scratch | **EduBlocks** | **Python**



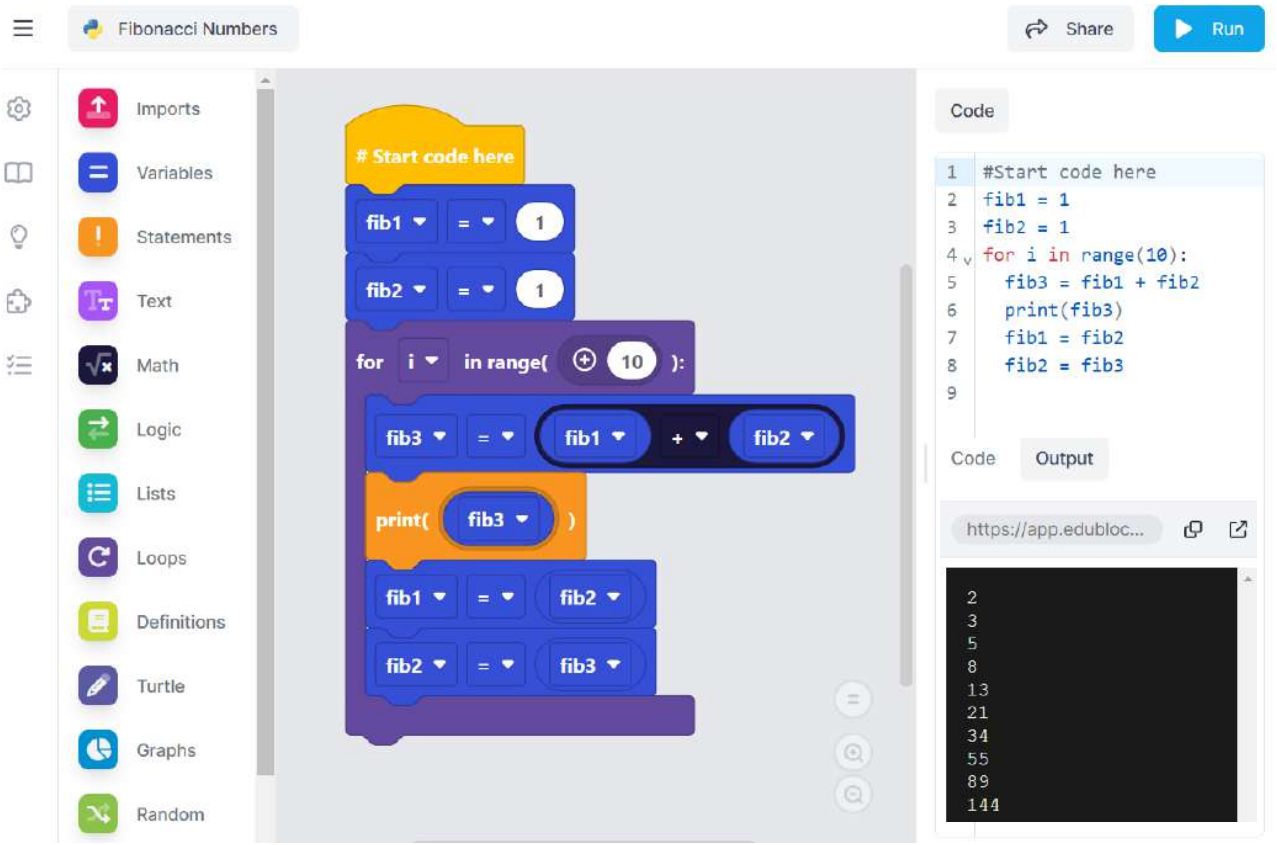
Εικόνα 2.3. Αλγόριθμος για τον σχεδιασμό τετραγώνου σε διαφορετικά περιβάλλοντα προγραμματισμού

Παραπάνω φαίνεται ο σχεδιασμός ενός τετραγώνου με πλευρά 100 σε Scratch, σε Edublocks και στη γλώσσα Python. Στο περιβάλλον Edublocks κάθε μπλοκ αντιστοιχεί σε μια εντολή της γλώσσας Python. Η διαφορά με το Scratch είναι ότι εδώ θα πρέπει πρώτα να δημιουργήσουμε το αντικείμενο, π.χ. τον franklin το χελωνάκι, στο οποίο θα δώσουμε τις κατάλληλες εντολές, για παράδειγμα να κινηθεί μπροστά (franklin.forward()) ή να στρίψει αριστερά (franklin.left()). Αυτού του είδους ο προγραμματισμός που είναι προσανατολισμένος ή, όπως λέγεται, στρέφεται προς το αντικείμενο λέγεται **αντικειμενοστρεφής.** Παρακάτω δίνεται μια εξήγηση για καθεμία από τις εντολές της Python που εμφανίζονται στο παραπάνω πρόγραμμα.

from turtle import *	Προσάρτηση της βιβλιοθήκης turtle την οποία θα χρησιμοποιήσουμε.
franklin = Turtle( )	Δημιουργία ενός αντικειμένου με όνομα franklin, τύπου Turtle.
franklin.pendown( )	Εντολή στον franklin να κατεβάσει την πένα, ώστε να αφήνει το ίχνος του.
franklin.forward(100)	Εντολή στον franklin να προχωρήσει μπροστά.
franklin.left(90)	Εντολή στον franklin να στρίψει αριστερά 90°.
for i in range(4):	Επανάλαβε 4 φορές

2.2.1 Το περιβάλλον EduBlocks

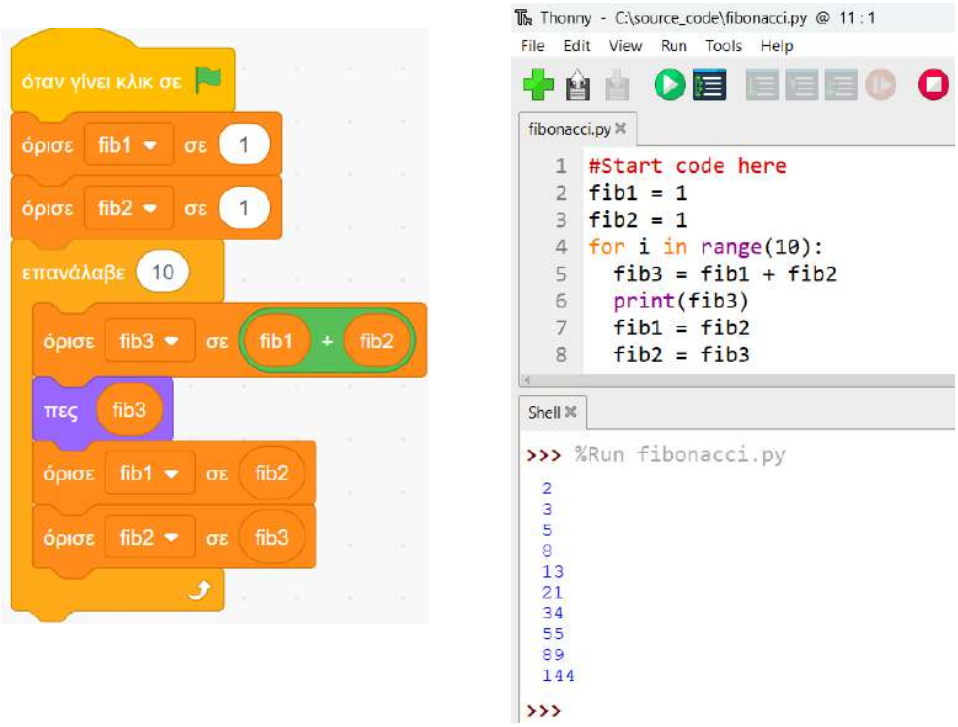
Το περιβάλλον προγραμματισμού που θα χρησιμοποιηθεί για την ανάπτυξη προγραμμάτων στη γλώσσα Python είναι το περιβάλλον οπτικού προγραμματισμού με πλακίδια (block-based) EduBlocks. Μπορούν όμως να χρησιμοποιηθούν και τα περιβάλλοντα κειμενικού προγραμματισμού IDLE και Thonny, αν προτιμάτε το γράψιμο κώδικα από τη μετακίνηση πλακιδίων (blocks).



Εικόνα 2.4. Αλγόριθμος υπολογισμού αριθμών Fibonacci στο περιβάλλον EduBlocks.

Το περιβάλλον EduBlocks έχει ομάδες εντολών σε μορφή πλακιδίων, όπως ακριβώς και το Scratch. Η διαφορά εδώ είναι ότι, εκτός από τον κώδικα στη μέση της οθόνης, δεξιά παράγεται και ο ισοδύναμος κώδικας σε Python. Για να εκτελέσουμε το πρόγραμμά μας, πατάμε πάνω δεξιά, εκεί που λέει RUN, και στη θέση της καρτέλας Code εμφανίζεται η καρτέλα Output, όπου εκτυπώνονται τα αποτελέσματα από την εντολή print. Κάθε μπλοκ ισοδυναμεί με μια γραμμή κώδικα σε γλώσσα Python. Το παραπάνω πρόγραμμα υπολογίζει τους πρώτους 11 αριθμούς Fibonacci. Η ακολουθία Fibonacci είναι μια ακολουθία αριθμών που εμφανίζεται στη φύση, όπως για παράδειγμα στη διάταξη των φύλλων ενός φυτού και στα πέταλα των λουλουδιών. Οι σπείρες των κοχυλιών και των σαλιγκαριών συχνά ακολουθούν την ακολουθία Fibonacci. Το αρχικό παράδειγμα που χρησιμοποίησε ο Fibonacci για να περιγράψει την ακολουθία του είναι η αναπαραγωγή κουνελιών. Κάθε ζεύγος

κουνελιών παράγει ένα νέο ζεύγος κάθε μήνα, με κάθε νέο ζεύγος να ξεκινά την αναπαραγωγή από τον δεύτερο μήνα.
Έτσι έχουμε τους αριθμούς 1, 1, 2, 3, 5, 8, 13, 21 κ.λπ. Όπως φαίνεται, κάθε αριθμός στην ακολουθία είναι το άθροισμα των δυο προηγουμένων. Ο αλγόριθμος υπολογισμού των αριθμών Fibonacci στο περιβάλλον προγραμματισμού Scratch και στο περιβάλλον προγραμματισμού Thonny φαίνεται στην Εικόνα 2.5.



Εικόνα 2.5. Αλγόριθμος υπολογισμού των αριθμών Fibonacci στο περιβάλλον προγραμματισμού Scratch και στο περιβάλλον προγραμματισμού Thonny.

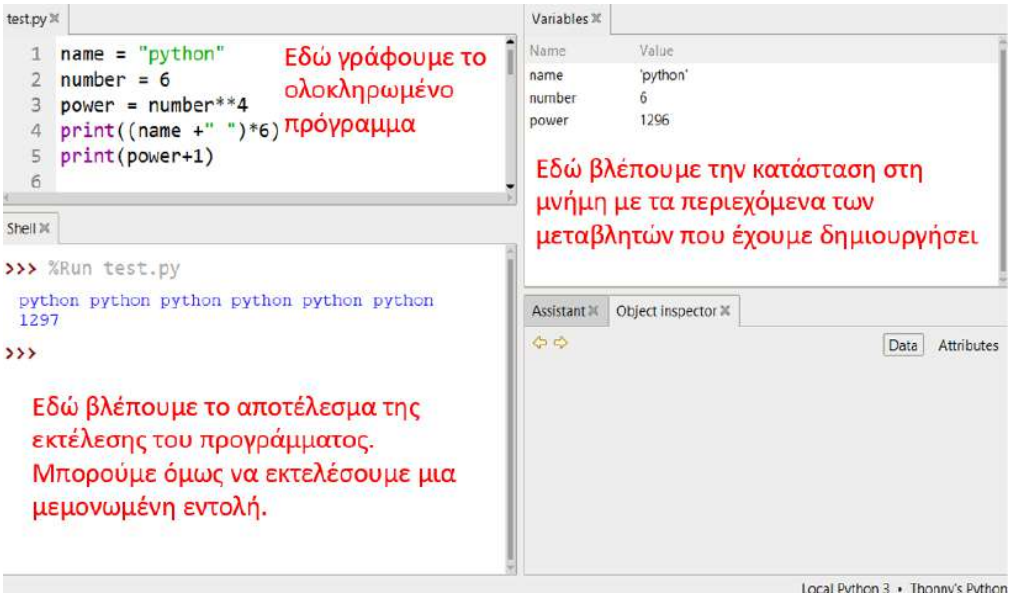
Στο περιβάλλον Thonny, κάτω από τον πηγαίο κώδικα σε Python, φαίνεται και το αποτέλεσμα της εκτέλεσής του, δηλαδή το αποτέλεσμα της εντολής print η οποία εμφανίζει την τιμή της μεταβλητής fib3 στην οθόνη.

Όταν αρχίσετε να γράφετε προγράμματα με πολλές γραμμές κώδικα, θα διαπιστώσετε ότι η διαχείρισή τους σε ένα περιβάλλον με πλακίδια δεν είναι τόσο εύκολη. Μπορείτε να μεταβείτε από το περιβάλλον EduBlocks στο περιβάλλον κειμενικού (textual) προγραμματισμού, όπου μπορείτε να πληκτρολογείτε τον κώδικά σας στη γλώσσα Python, όποτε το κρίνεται εσείς.

Το πιο απλό περιβάλλον προγραμματισμού για Python είναι το IDLE, το οποίο μπορείτε να κατεβάσετε από την ιστοσελίδα της Python https://www.python.org/. Η Python είναι μια γλώσσα που αναπτύσσεται συνεχώς και προσαρμόζεται στις εξελίξεις στην έρευνα και τη βιομηχανία. Η πιο πρόσφατη έκδοσή της είναι η 3.12.4.

2.2.2 Το περιβάλλον προγραμματισμού Thonny

Το Thonny είναι ένα δωρεάν και ανοικτού κώδικα περιβάλλον προγραμματισμού για τη γλώσσα Python. Μπορείτε να το κατεβάσετε από τη διεύθυνση: https://thonny.org/.



Εικόνα 2.6. Το περιβάλλον προγραμματισμού Thonny

Εκτός από το παράθυρο του συντάκτη κειμένου που βρίσκεται πάνω αριστερά μπορούμε να δοκιμάσουμε και εντολές στο τερματικό του διερμηνευτή, αν θέλουμε να πειραματιστούμε και να δούμε τι λειτουργία επιτελούν. Εναλλακτικά περιβάλλοντα που μπορείτε να χρησιμοποιήσετε αντί για το Thonny είναι το IDLE (https://www.python.org/) και το Spyder (https://www.spyder-ide.org/).

2.3 Τύποι δεδομένων

Κάθε γλώσσα προγραμματισμού αποθηκεύει τα δεδομένα στη μνήμη σε διάφορες μορφές. Στο βιβλίο αυτό θα διαχωρίσουμε τα δεδομένα σε δυο βασικές κατηγορίες: τις λέξεις και τους αριθμούς. Οι λέξεις στην ορολογία της Πληροφορικής λέγονται αλφαριθμητικά, επειδή μπορούν να περιέχουν γράμματα και αριθμούς, όπως για παράδειγμα ο κωδικός πρόσβασής σας (password) στην ηλεκτρονική τάξη. Τα **αλφαριθμητικά** είναι μια ακολουθία από χαρακτήρες που μπορούν να είναι ψηφία, γράμματα ή σημεία στίξεως και βρίσκονται μέσα σε εισαγωγικά (quotes) (διπλά ή μονά). Οι **αριθμητικοί τύποι** στην Python είναι οι ακέραιοι (integer) (**int**) και οι πραγματικοί (**float**) αριθμοί. Για την υποδιαστολή στον προγραμματισμό χρησιμοποιείται η τελεία «.» και όχι το κόμμα.

Τα δεδομένα μπορεί να είναι αριθμοί, ονόματα ή αποτελέσματα λογικών αποφάσεων. Στην Python έχουμε τους εξής βασικούς τύπους:

Τύπος	Ονομασία	Παραδείγματα
Ακέραιοι	int	1, 0, –1, 496
Πραγματικοί	float	3.14159, 0.5, 3.0
Λογικές	bool	True, False
Αλφαριθμητικά	str	“SARS-CoV-2”, “Ζάννειο Γυμνάσιο”
Λίστες	list	[6, 28, 496], [“Γη”, “Σελήνη”, “Ήλιος”]

Οι **λίστες** είναι ο σημαντικότερος σύνθετος τύπος της Python. Μια λίστα είναι μια ακολουθία από αντικείμενα οποιουδήποτε τύπου. Οι λίστες είναι δυναμικές δομές, δηλαδή μπορούμε να προσθέσουμε ή να αφαιρέσουμε στοιχεία αυξομειώνοντας το μέγεθός τους.
Η Python μας δίνει τη δυνατότητα να εκτελέσουμε διάφορες πράξεις στα δεδομένα, χρησιμοποιώντας τους αντίστοιχους τελεστές. Οι τελεστές (operators) είναι σύμβολα ή λέξεις για τη δημιουργία αριθμητικών και λογικών εκφράσεων. Οι βασικότερες κατηγορίες τελεστών είναι οι αριθμητικοί, οι συγκριτικοί και οι λογικοί. Υπάρχουν όμως και τελεστές για πράξεις σε αλφαριθμητικά δεδομένα και σε λίστες αντίστοιχοι με τους αριθμητικούς τελεστές.

Αριθμητικοί τελεστές: είναι τα σύμβολα που χρησιμοποιούμε για να κάνουμε μαθηματικές πράξεις. Ο υπολογισμός κάθε έκφραση στην οποία υπάρχουν αριθμητικοί τελεστές ακολουθεί μια αυστηρά καθορισμένη ιεραρχία πράξεων, που είναι: **1.** Ύψωση σε δύναμη. **2.** Πολλαπλασιασμός, διαίρεση. **3.** Πρόσθεση, αφαίρεση.	Πρόσθεση Αφαίρεση Πολλαπλασιασμός Διαίρεση Πηλίκο Ακέραιας Διαίρεσης Ύψωση σε δύναμη Υπόλοιπο ακέραιας διαίρεσης	**+** **–** ***** **/** **//** ****** **%**

Αν θέλουμε να αλλάξουμε την ιεραρχία των πράξεων, μπορούμε να χρησιμοποιήσουμε παρενθέσεις. Για παράδειγμα, στην έκφραση (200+1)*10 θα εκτελεστεί πρώτα η πρόσθεση μέσα στην παρένθεση και μετά το αποτέλεσμα θα πολλαπλασιαστεί επί 10, που μας κάνει 2010, σε αντίθεση με την έκφραση 200+1*10, στην οποία πρώτα θα γίνει ο πολλαπλασιασμός και μετά η πρόσθεση, άρα θα πάρουμε 210.

Οι **Σχεσιακοί** (ή συγκριτικοί) τελεστές χρησιμοποιούνται για τη σύγκριση εκφράσεων. Αν η σχέση ισχύει το αποτέλεσμα είναι Αληθής (True) αλλιώς Ψευδής (False). Μπορούμε να τις αντιμετωπίζουμε ως ερωτήσεις που απαντώνται με ΝΑΙ ή ΟΧΙ.
`>>> 496==496` `>>> 12<11 and 23>10`
`True` `False`
`>>> 12!=13` `>>> 12<11 or 23>10` `>>> not(56<=12)`
`True` `True` `True`
	Μικρότερο από
	Μικρότερο ή ίσο από
	Μεγαλύτερο από
	Μεγαλύτερο ή ίσο από
	Ίσο με
	Διάφορο από
		<
		<=
		>
		>=
		==
		!=

Οι **Λογικοί** τελεστές χρησιμοποιούνται όταν θέλουμε να σχηματίσουμε σύνθετες συνθήκες. Η σύζευξη λογικών εκφράσεων είναι Αληθής, αν και μόνον αν είναι αληθείς όλες οι εκφράσεις, ενώ η διάζευξη είναι αληθής, αν συμμετέχει σε αυτήν τουλάχιστον μια αληθής έκφραση.	Άρνηση (ΟΧΙ) Σύζευξη (ΚΑΙ) Διάζευξη (Η)	**not** **and** **or**

Μπορούμε να δοκιμάσουμε εντολές ή να υπολογίσουμε εκφράσεις χρησιμοποιώντας τον διερμηνευτή της Python, ο οποίος μεταφράζει σε γλώσσα μηχανής και εκτελεί μια μεμονωμένη εντολή χωρίς να χρειαστεί να αναπτύξουμε ολόκληρο πρόγραμμα. Ο τελεστής (συνάρτηση) str μετατρέπει ένα αντικείμενο άλλου τύπου σε αλφαριθμητικό. Ο τελεστής της πρόσθεσης «+» όταν εφαρμόζεται σε αλφαριθμητικά, εκτελεί συνένωση ενώ αυτός του πολλαπλασιασμού «*» παράγει τη συμβολοσειρά επαναλαμβανόμενη τόσες φορές, όσες είναι ο αριθμός. Ο τελεστής int μετατρέπει ένα αντικείμενο σε ακέραιο αριθμό, εάν αυτό είναι εφικτό.

```
>>> 10+10
20
>>> "10"+"10"
'1010'
>>> 5*"10"
'1010101010'
>>> 5*10
50
>>> 5*int("10")
50
>>> 5*str(10)
'1010101010'
```

```
>>> number = input("Δώσε έναν αριθμό = ")
 Δώσε έναν αριθμό = 28
>>> print(number+number+number)
 282828
>>> print(3*number)
 282828
>>> number = int( input("Δώσε έναν αριθμό = ") )
 Δώσε έναν αριθμό = 28
>>> print(number+number+number)
 84
>>> print(3*number)
 84
```

Οι τελεστές μετατροπής τύπων int, float και str χρειάζονται κατά την εισαγωγή δεδομένων από τον χρήστη ή από άλλη πηγή, για παράδειγμα από κάποιο αρχείο. Ακόμα και αριθμό να δώσει ο χρήστης, η Python θα το εκλάβει ως αλφαριθμητικό, οπότε θα χρειαστεί να το μετατρέψουμε στην κατάλληλη μορφή. Στο παραπάνω παράδειγμα ο χρήστης δίνει τον αριθμό 28, τον οποίο διαβάζει η εντολή input. Στην πρώτη περίπτωση αντιμετωπίζεται σαν αλφαριθμητικό και στη δεύτερη σαν ακέραιος αριθμός.

Δραστηριότητα 1

Μεταβείτε στο κέλυφος (shell) του Thonny και δώστε τις παρακάτω εκφράσεις:

```
>>> 2**10    >>> 2**100    >>> 2**1000    >>> 2**10000
```

Δραστηριότητα 2

Μεταβείτε στο κέλυφος (shell) του Thonny και δώστε τις παρακάτω εκφράσεις:

```
>>> 5*("Grace" + "Hopper" + " , ")    >>> 5*"Grace" + 5*"Hopper" + 5*" , "    >>> "Alan Turing"[5:10]
```

Στη συνέχεια μεταβείτε στο ChatGPT ή στο Gemini και δώστε την προτροπή:
Μπορείς να μου εξηγήσεις τη λειτουργία που επιτελεί η έκφραση "python"[2:4]

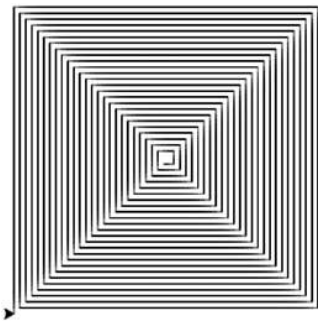
Δραστηριότητα 3

Μεταβείτε στο κέλυφος (shell) του Thonny και δώστε τις παρακάτω εκφράσεις:

```
>>> len(str(8128))    >>> len(str(496))    >>> len(str(10**20))    >>> len(str(2**100))    >>> str(len(5))
```

Τι κάνει η συνάρτηση len και τι η str; Τι πετυχαίνουμε με τη συνδυασμένη χρήση τους;
Στη συνέχεια μεταβείτε στο ChatGPT ή στο Gemini και δώστε την προτροπή:
Ποια λειτουργία επιτελεί η len(str(number));

2.4 Μεταβλητές



Για να αποθηκεύσουμε τα δεδομένα στη μνήμη του υπολογιστή, χρησιμοποιούμε τις **μεταβλητές**. Οι μεταβλητές είναι συμβολικά ονόματα που αναφέρονται σε δεδομένα τα οποία βρίσκονται σε κάποια θέση στη μνήμη του υπολογιστή. Κάθε φορά στη θέση αυτή μπορεί να αποθηκευτεί μόνο μία τιμή. Η χρήση μεταβλητών είναι αναγκαία για την επίλυση κάποιων προβλημάτων, όπως για παράδειγμα ο σχεδιασμός ενός σπιράλ όπως είχαμε δει στην Α΄ Γυμνασίου. Στην περίπτωση αυτή η μεταβλητή εκφράζει το μεταβλητό μήκος της πλευράς που αυξάνεται σε κάθε επανάληψη.

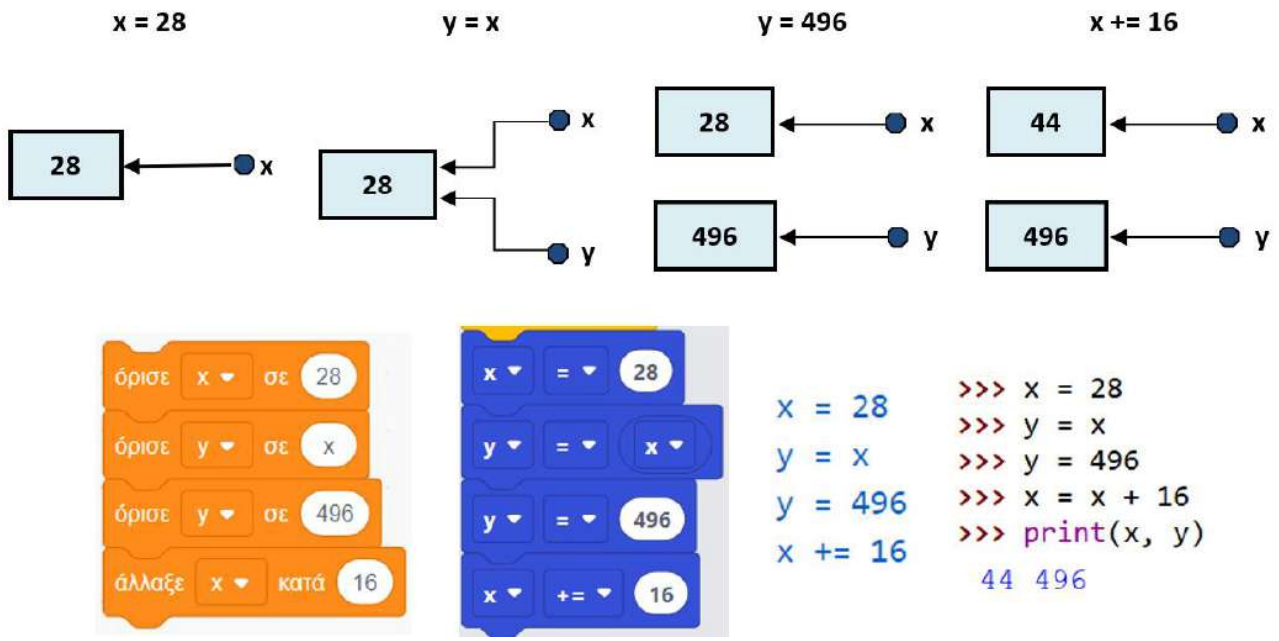
Παρακάτω υπενθυμίζουμε το πρόγραμμα που σχεδιάζει ένα σπιράλ σε Scratch και στο περιβάλλον EduBlocks με τη μορφή πλακιδίων και με τη μορφή κώδικα Python.

Εικόνα 2.5. Πρόγραμμα σχεδιασμού ενός σπιράλ σε διαφορετικά περιβάλλοντα προγραμματισμού

Η διαφορά είναι ότι, ενώ στο Scratch έχουμε ήδη δημιουργήσει ένα αντικείμενο στο σκηνικό, στο EduBlocks το αντικείμενο δημιουργείται από τον κατασκευαστή αντικειμένων, που στην περίπτωση της χελώνας είναι ο κατασκευαστής Turtle( ). Στη συνέχεια δίνουμε στο αντικείμενο franklin τις κατάλληλες εντολές. Έτσι, στον ίδιο κώδικα μπορούμε να έχουμε περισσότερα από ένα αντικείμενα, ενώ ο κώδικας σε Scratch αναφέρεται σε συγκεκριμένο αντικείμενο. Η μεταβλητή length ξεκινάει με τιμή 10 και αυξάνεται κατά 3 κάθε φορά.

Στη γλώσσα Python υπάρχουν ορισμένοι κανόνες που πρέπει να ακολουθούμε σχετικά με το όνομα μιας μεταβλητής. Έτσι, για παράδειγμα, το όνομα μιας μεταβλητής δεν επιτρέπεται να ξεκινά με αριθμό, ούτε πρέπει να

είναι όμοιο με κάποιο όνομα ενσωματωμένης συνάρτησης ή εντολής ή άλλης μεταβλητής. Καλό είναι να δίνουμε στις μεταβλητές ονόματα ενδεικτικά της σημασίας των δεδομένων που καταχωρούνται σε αυτές.
Για να δώσουμε τιμή σε μια μεταβλητή, χρησιμοποιούμε τον τελεστή ίσον «=», για παράδειγμα *grade = 8.51*. Διαβάζεται «Το grade να γίνει 8.51». Πάντα στο αριστερό μέλος μια εντολής απόδοσης τιμής πρέπει να υπάρχει μια μεταβλητή, ενώ στο δεξί μέλος μπορεί να υπάρχει οποιαδήποτε έκφραση, έτσι ώστε η εντολή να έχει νόημα. Η εντολή x = 28 σημαίνει ότι η μεταβλητή με όνομα x αναφέρεται σε μια θέση στη μνήμη με περιεχόμενο 28. Αν στη συνέχεια θέσουμε y = x, τότε και η μεταβλητή y αναφέρεται στην τιμή 28. Όταν θέσουμε στη συνέχεια y = 496, η μεταβλητή y αναφέρεται στην τιμή 496.



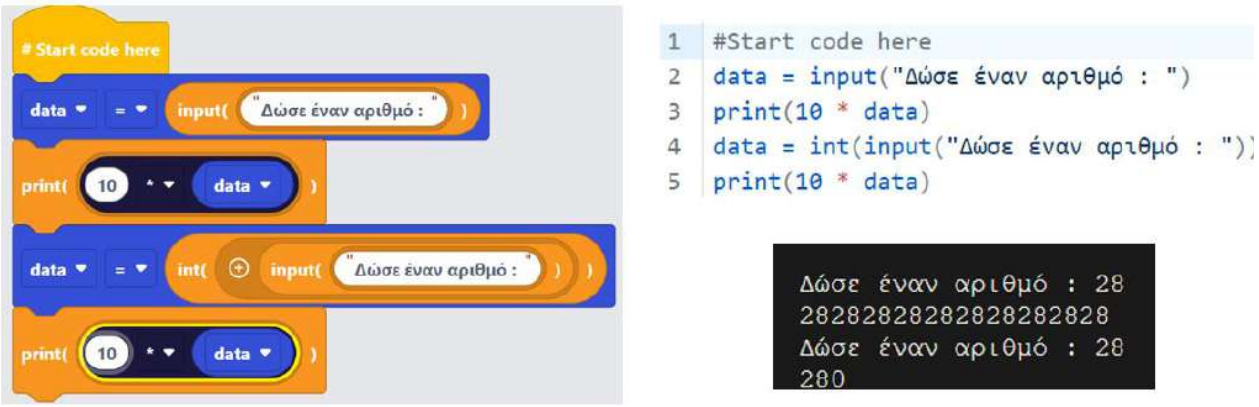
Εικόνα 2.6: Εντολές απόδοσης τιμής μεταξύ μεταβλητών σε Scratch, EduBlocks και Thonny

Αναπαράσταση μεταβλητών στη γλώσσα Python

Στις σύγχρονες γλώσσες προγραμματισμού η μεταβλητή δεν είναι μια θέση στη μνήμη με περιεχόμενο την τιμή που της έχει αποδοθεί, αλλά δείχνει σε μια θέση στη μνήμη στην οποία βρίσκεται αυτή η τιμή. Αυτό σημαίνει ότι, αν δυο μεταβλητές έχουν την τιμή 28, δεν υπάρχουν δύο θέσεις μνήμης με την τιμή 28, αλλά μια θέση μνήμης στην οποία αναφέρονται και οι δυο μεταβλητές.

2.5 Εντολές εισόδου δεδομένων – εξόδου αποτελεσμάτων

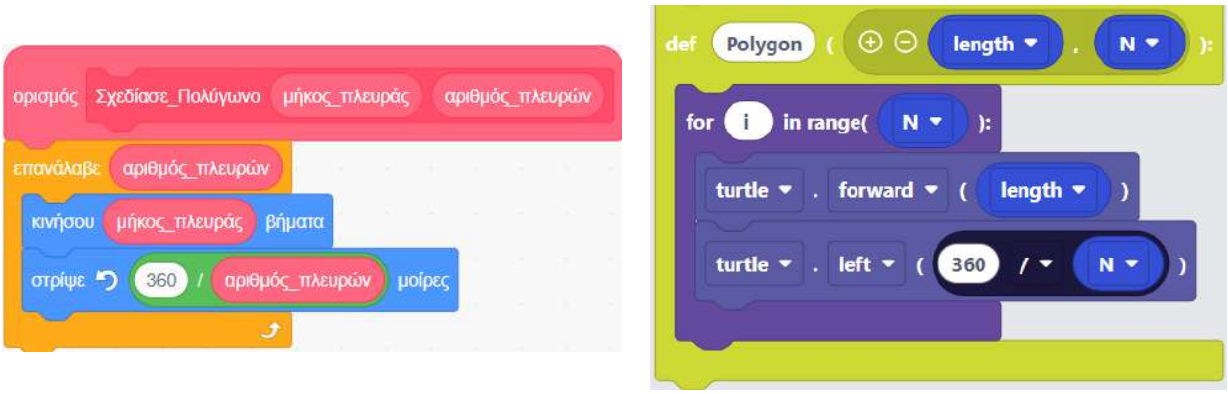
Η εντολή print εμφανίζει μηνύματα και το περιεχόμενο μεταβλητών στην οθόνη. Τη χρησιμοποιούμε αν θέλουμε να εμφανίσουμε τα αποτελέσματα ενός προγράμματος. Αντίθετα η εντολή input δέχεται από τον χρήστη δεδομένα σε μορφή κειμένου. Ανάλογα με τον τύπο των δεδομένων κάποιες φορές χρειάζεται η κατάλληλη μετατροπή. Στο παρακάτω παράδειγμα αρχικά το 28 εκλαμβάνεται ως αλφαριθμητικό, γι' αυτό επαναλαμβάνονται τα ψηφία του δέκα φορές. Στη δεύτερη περίπτωση τα δεδομένα εισόδου μετατρέπονται με την int σε ακέραια τιμή, οπότε ο τελεστής «*» εκτελεί πολλαπλασιασμό και όχι διαδοχικές συνενώσεις.



Εικόνα 2.7. Παράδειγμα εντολής εισόδου δεδομένων και εξόδου αποτελεσμάτων

2.6 Υποπρογράμματα - Συναρτήσεις

Σε προηγούμενη τάξη σχεδιάσαμε ένα υποπρόγραμμα για τον σχεδιασμό ενός πολυγώνου με παραμέτρους τον αριθμό και το μήκος των πλευρών στο Scratch. Παρακάτω φαίνεται το ίδιο υποπρόγραμμα στο EduBlocks.



Εικόνα 2.8. Υποπρόγραμμα για το σχεδιασμό πολυγώνου

Ο κώδικας στη γλώσσα Python φαίνεται δίπλα. Παρατηρήστε ότι οι εντολές που βρίσκονται εντός της εντολής επανάληψης *for* βρίσκονται μια εσοχή πιο δεξιά.

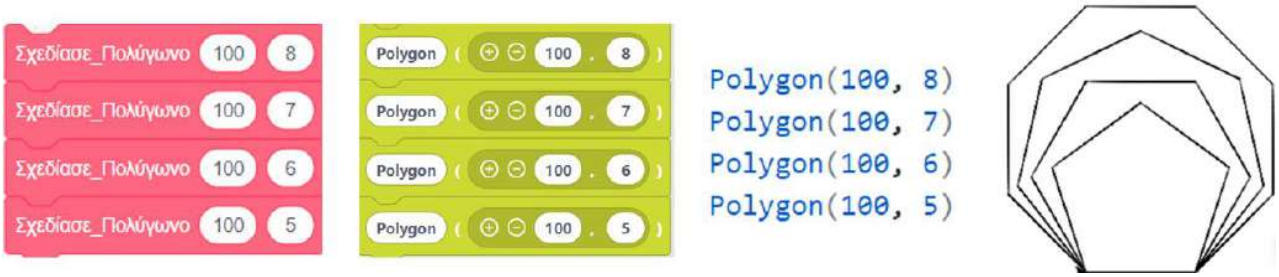
```
def Polygon(length, N):
  for i in range(N):
    turtle.forward(length)
    turtle.left(360 / N)
```

Ποια είναι η λειτουργία της εντολής *for*;

Για να χρησιμοποιήσουμε το υποπρόγραμμα που ορίσαμε, το καλούμε με το όνομά του, όπως κάνουμε και με οποιαδήποτε άλλη εντολή, δίνοντας κατάλληλες τιμές στις παραμέτρους του. Για παράδειγμα, αν θέλουμε να σχεδιαστεί ένα εξάγωνο πλευράς 100, θα δώσουμε την εντολή Polygon(100, 6) στην Python.

Στην Python τα υποπρογράμματα λέγονται και **συναρτήσεις** και ορίζονται με τη δεσμευμένη λέξη **def**.

Παρακάτω δίνουμε τις εντολές για το σχεδιασμό ενός οκταγώνου, ενός επταγώνου, ενός εξαγώνου και ενός πενταγώνου.



Εικόνα 2.9. Κλήση υποπρογραμμάτων για το σχεδιασμό πολυγώνων

Δραστηριότητα 1

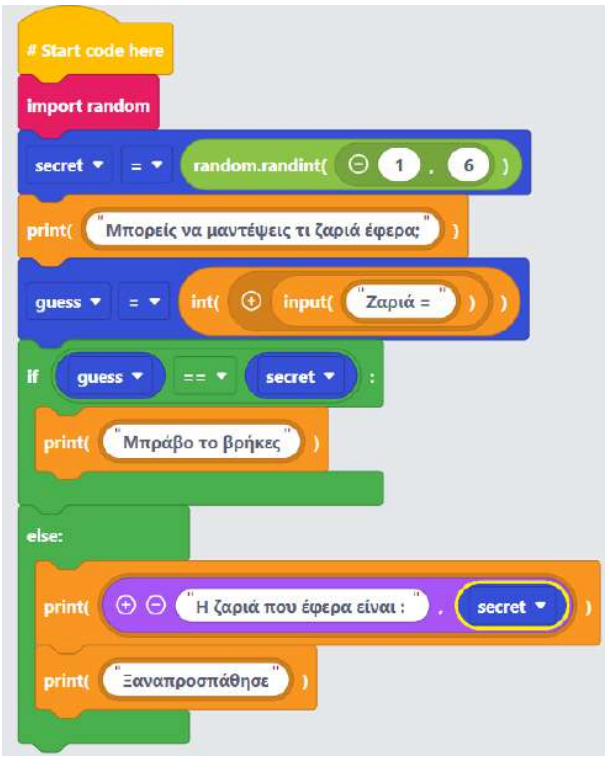
Να γράψετε ένα πρόγραμμα σε Python το οποίο θα σχεδιάζει 12 πολύγωνα, ξεκινώντας από ένα τετράγωνο και καταλήγοντας σε ένα δεκαπεντάγωνο, αυξάνοντας το πλήθος των πλευρών κατά 1 κάθε φορά, γράφοντας την εντολή Polygon μόνο μια φορά.

Δραστηριότητα 2

Να γράψετε ένα πρόγραμμα σε Python το οποίο θα σχεδιάζει 20 τετράγωνα το ένα μέσα στο άλλο. Κάθε τετράγωνο θα έχει πλευρά μεγαλύτερη κατά 10 από το προηγούμενο.

2.7 Η τύχη βοηθάει τους τολμηρούς

Ένα βασικό χαρακτηριστικό των υπολογιστών είναι ο **ντετερμινισμός (determinism)**, δηλαδή η βεβαιότητα ότι όσες φορές και να εκτελέσουμε ένα πρόγραμμα, αυτό θα καταλήγει πάντα στο ίδιο αποτέλεσμα για την ίδια είσοδο. Έτσι όμως δε θα ήταν εφικτό να σχεδιαστούν τα βιντεοπαιχνίδια που ξέρουμε, γιατί όλες οι κινήσεις του υπολογιστή θα ήταν προβλέψιμες. Γι' αυτό υπάρχει ένας μηχανισμός παραγωγής τυχαίων αριθμών. Στη γλώσσα Python αυτό υλοποιείται με τη συνάρτηση randint της βιβλιοθήκης random.
Γράφοντας random.randint (1, 6) παράγεται ένας ακέραιος αριθμός από 1 έως και 6 με τυχαίο τρόπο. Η εντολή import random ενημερώνει τον διερμηνευτή ότι θα χρησιμοποιηθεί η βιβλιοθήκη random.

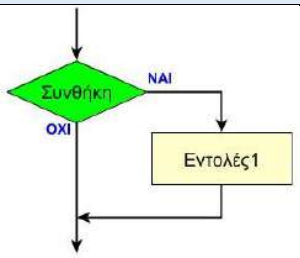



```
1  #Start code here
2  import random
3  secret = random.randint(1, 6)
4  print("Μπορείς να μαντέψεις τι ζαριά έφερα;")
5  guess = int(input("Ζαριά = "))
6  if guess == secret:
7    print("Μπράβο το βρήκες")
8  else:
9    print("Η ζαριά που έφερα είναι : ", secret)
10   print("Ξαναπροσπάθησε")
11
```

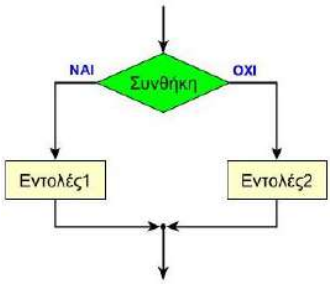
Στο παράδειγμα αυτό ο υπολογιστής δίνει την ψευδαίσθηση ότι σκέφτεται έναν αριθμό. Κάθε φορά επιλέγει και διαφορετικό τυχαίο αριθμό, και ο χρήστης προσπαθεί να μαντέψει τον αριθμό που σκέφτηκε. Γι' αυτό χρειαζόμαστε έναν έλεγχο αν ο αριθμός που σκέφτηκε ο υπολογιστής είναι ίδιος με τον αριθμό που μάντεψε ο άνθρωπος. Γι' αυτό χρησιμοποιείται η εντολή if guess == secret. Αν η συνθήκη ισχύει, σημαίνει ότι ο χρήστης μάντεψε σωστά (βρήκε τον αριθμό που «σκέφτηκε» ο υπολογιστής). Αν η συνθήκη δεν ισχύει, το πρόγραμμα εμφανίζει κάποιο άλλο μήνυμα.

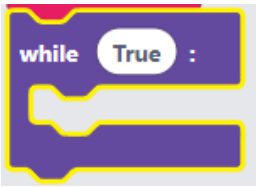
Εικόνα 2.10. Παράδειγμα προγράμματος με χρήση του μηχανισμού παραγωγής τυχαίων αριθμών

Αυτό το είδος λειτουργίας στο πλαίσιο ενός προγράμματος ονομάζεται **εκτέλεση υπό συνθήκη** (conditional execution). Γι' αυτό χρησιμοποιούμε την εντολή if. Η εντολή if μπορεί να χρησιμοποιηθεί στις δυο παρακάτω βασικές μορφές:

Απλή δομή επιλογής		
Αν ισχύει η Συνθήκη Τότε Εντολές1 Τέλος_Αν	if <Συνθήκη>: Εντολές1 # Εκτελούνται όταν η συνθήκη ισχύει (True), αλλιώς παραλείπονται	

Σύνθετη δομή επιλογής		

Αν ισχύει η Συνθήκη Τότε
Εντολές1
Αλλιώς
Εντολές2
Τέλος_Αν
	if <Συνθήκη>:
	Εντολές1
	else:
	Εντολές2
		



Το παραπάνω παιχνίδι μεταξύ ανθρώπου και υπολογιστή διαρκεί όμως μόνο ένα γύρο. Θα μπορούσαμε με την προσθήκη μιας μόνο εντολής να δώσουμε περισσότερες ευκαιρίες στον άνθρωπο να μαντέψει τον αριθμό, θέτοντας όλες τις εντολές εντός του μπλοκ της εντολής επανάληψης while True, δηλαδή μιας εντολής που εκτελείται όσο η έκφραση True είναι Αληθής, άρα για πάντα.

Τώρα ο παίκτης μπορεί να παίξει πολλούς γύρους.

Η εντολή με την οποία σκέφτεται ο υπολογιστής, δηλαδή παράγεται ένας τυχαίος αριθμός, βρίσκεται εκτός της εντολής επανάληψης, άρα εκτελείται μόνο μια φορά. Αυτό έχει ως αποτέλεσμα ο παίκτης να μπορεί να βρει την απάντηση, κάνοντας το πολύ έξι προσπάθειες. Μια λύση στο πρόβλημα αυτό είναι να σκέφτεται ο υπολογιστής διαφορετικό αριθμό κάθε φορά.

```
import random
secret = random.randint(1, 6)
while True:
  print("Μπορείς να μαντέψεις τι ζαριά έφερα;")
  guess = int(input("Ζαριά = "))
  if guess == secret:
    print("Μπράβο το βρήκες")
  else:
    print("Ξαναπροσπάθησε")
```

Ακόμα όμως και αν ο παίκτης βρει την απάντηση, η εντολή επανάληψης δε θα σταματήσει, αφού εκτελείται για πάντα. Γι' αυτό μπορούμε, αμέσως μόλις ο παίκτης βρει την απάντηση, να διακόψουμε την επανάληψη με χρήση της εντολής break. Αν δε θέλουμε να χρησιμοποιήσουμε την εντολή break, μπορούμε να χρησιμοποιήσουμε μια λογική μεταβλητή η οποία θα είναι αρχικά False και μόλις ο παίκτης βρει τη σωστή απάντηση θα γίνει True. Άρα τώρα η εντολή επανάληψης διατυπώνεται ως «Όσο η μεταβλητή Found δε γίνεται True συνέχισε».

```python
import random
while True:
  secret = random.randint(1, 6)
  print("Μπορείς να μαντέψεις τι ζαριά έφερα;")
  guess = int(input("Ζαριά = "))
  if guess == secret:
    print("Μπράβο το βρήκες")
    break
  else:
    print("Ξαναπροσπάθησε")
```

```python
import random
Found = False
while not Found:
  secret = random.randint(1, 6)
  print("Μπορείς να μαντέψεις τι ζαριά έφερα;")
  guess = int(input("Ζαριά = "))
  if guess == secret:
    print("Μπράβο το βρήκες")
    Found = True
  else:
    print("Ξαναπροσπάθησε")
```

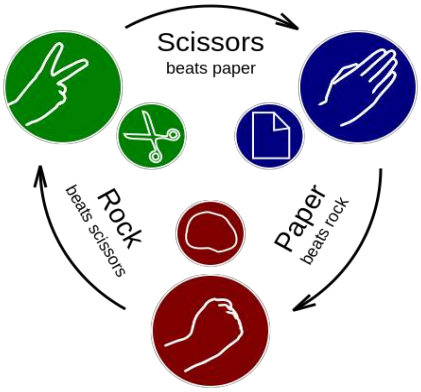
Ποια από τις παραπάνω λύσεις φαίνεται πιο απλή και επεξηγηματική;

2.8 Παίζοντας Πέτρα-Ψαλίδι-Χαρτί με τον υπολογιστή

Ένα δημοφιλές παιχνίδι το οποίο μπορούμε να παίξουμε με τον υπολογιστή είναι το Πέτρα-Ψαλίδι-Χαρτί. Οι δυο παίκτες, ο υπολογιστής και ο άνθρωπος, επιλέγουν ένα εκ των τριών αντικειμένων είτε στην τύχη είτε με κάποια στρατηγική. Οι κανόνες είναι ότι :

- Η πέτρα κερδίζει το ψαλίδι
- Το ψαλίδι κερδίζει το χαρτί
- Το χαρτί κερδίζει την πέτρα

Όταν επιλεγεί το ίδιο αντικείμενο έχουμε ισοπαλία.

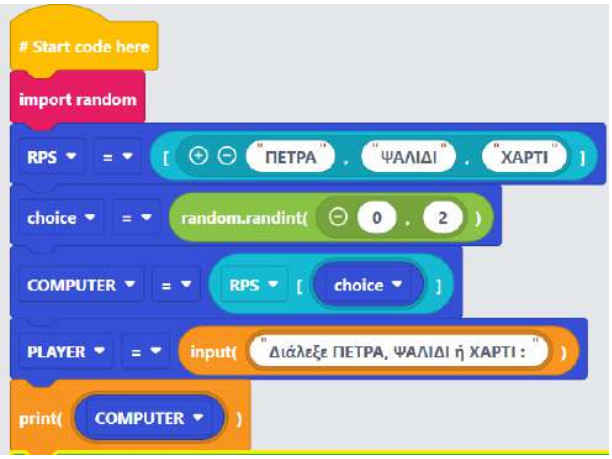


Εικόνα 2.11. Σχηματική αναπαράσταση των κανόνων του παιχνιδιού Πέτρα-Ψαλίδι-Χαρτί

Το παρακάτω πρόγραμμα σε Python, που έχει αναπτυχθεί στο περιβάλλον με πλακίδια EduBlocks, παίζει το παιχνίδι για ένα γύρο. Συγκρίνεται η επιλογή του υπολογιστή με την είσοδο του χρήστη και με βάση τους κανόνες εμφανίζεται ο νικητής.

Σε μια λίστα RPS αποθηκεύονται οι τρεις επιλογές που έχουμε.

RPS	"ΠΕΤΡΑ"	"ΨΑΛΙΔΙ"	"ΧΑΡΤΙ"
	0	1	2



Εικόνα 2.12. Πρόγραμμα σε Python για το παιχνίδι Πέτρα-Ψαλίδι-Χαρτί (1ος γύρος)

Η συνάρτηση randint επιστρέφει με τυχαίο τρόπο έναν εκ των αριθμών 0,1 ή 2. Αν η randint επιστρέψει 0 τότε το RPS[0] = "ΠΕΤΡΑ", αν επιστρέψει 1 τότε RPS[1] = "ΨΑΛΙΔΙ", ενώ για 2 RPS[2] = "ΧΑΡΤΙ".

Αν choice = 0 ➔ RPS[choice] = RPS[0] = "ΠΕΤΡΑ"

Αν choice = 1 ➔ RPS[choice] = RPS[1] = "ΨΑΛΙΔΙ"

Αν choice = 2 ➔ RPS[choice] = RPS[2] = "ΧΑΡΤΙ"

} RPS[choice] είναι η επιλογή του υπολογιστή σε κάθε περίπτωση

Μελετώντας τον αντίστοιχο κώδικα σε Python παρατηρούμε ότι, όσο αυξάνεται το μέγεθος του προγράμματος, τόσο πιο δύσκολη γίνεται η διαχείρισή του στην αναπαράσταση με πλακίδια. Παρατηρούμε ότι η εντολή που εκτελείται όταν κερδίζει ο παίκτης επαναλαμβάνεται τρεις φορές. Αν προσθέσουμε και μια μεταβλητή wins για να καταγράφουμε το σκορ, η wins θα πρέπει να αυξάνεται κατά 1 με την εντολή wins+=1 για κάθε τέτοια περίπτωση.

```
import random
RPS = ["ΠΕΤΡΑ", "ΨΑΛΙΔΙ", "ΧΑΡΤΙ"]
choice = random.randint(0, 2)
computer = RPS[choice]
player = input("Διάλεξε ΠΕΤΡΑ, ΨΑΛΙΔΙ ή ΧΑΡΤΙ : ")
print(computer)
if player == "ΠΕΤΡΑ" and computer == "ΨΑΛΙΔΙ":
  print("ΚΕΡΔΙΣΕΣ ΜΠΡΑΒΟ")
elif player == "ΨΑΛΙΔΙ" and computer == "ΧΑΡΤΙ":
  print("ΚΕΡΔΙΣΕΣ ΜΠΡΑΒΟ")
elif player == "ΧΑΡΤΙ" and computer == "ΠΕΤΡΑ":
  print("ΚΕΡΔΙΣΕΣ ΜΠΡΑΒΟ")
elif player == computer:
  print("ΙΣΟΠΑΛΙΑ")
else:
  print("ΣΕ ΚΕΡΔΙΣΑ")
```

Για να μην επαναλαμβάνονται οι εντολές που εκτελούνται όταν κερδίζει ο παίκτης, μπορούμε να κατασκευάσουμε μια σύνθετη συνθήκη που θα περιλαμβάνει όλους τους συνδυασμούς νίκης του παίκτη.

Επίσης προσθέτουμε και δυο μεταβλητές (wins, losses) που προσμετρούν τις νίκες και ήττες από την πλευρά του παίκτη και μια μεταβλητή games που μετράει πόσα παιχνίδια έχουν γίνει.

Αρχικά πρέπει οι μεταβλητές να ξεκινήσουν από την τιμή 0. Η μεταβλητή games αυξάνεται σε κάθε παιχνίδι και, μόλις γίνει 4, η συνθήκη games<4 της while γίνεται False. Τότε η επανάληψη σταματάει και εκτελείται η επόμενη εντολή μετά την εντολή while, η οποία εμφανίζει το τελικό σκορ.

```
import random
RPS = ["ΠΕΤΡΑ", "ΨΑΛΙΔΙ", "ΧΑΡΤΙ"]
wins = losses = games = 0
while games<4 :
    games += 1
    choice = random.randint(0, 2)
    computer = RPS[choice]
    player = input("Διάλεξε ΠΕΤΡΑ, ΨΑΛΙΔΙ ή ΧΑΡΤΙ : ")
    print(computer)
    if ( player=="ΠΕΤΡΑ"  and computer=="ΨΑΛΙΔΙ" or
         player=="ΨΑΛΙΔΙ" and computer=="ΧΑΡΤΙ"  or
         player=="ΧΑΡΤΙ"  and computer=="ΠΕΤΡΑ" ) :
      print("ΚΕΡΔΙΣΕΣ ΜΠΡΑΒΟ")
      wins += 1
    elif player==computer:
      print("ΙΣΟΠΑΛΙΑ")
    else:
      print("ΣΕ ΚΕΡΔΙΣΑ")
      losses+=1
print(wins, " - ", losses)
```

Αντί για την εντολή while μπορεί να γίνει χρήση της εντολής for και έτσι να γλιτώσουμε τις γραμμές κώδικα της αρχικοποίησης της μεταβλητής games και της αύξησής της. Αντί για while games<4 μπορούμε να γράψουμε:

`for games in range(4)` που ισοδυναμεί με την εντολή `for games in [0, 1, 2, 3]`

Η συνάρτηση range επιστρέφει μια λίστα με τους αριθμούς 0,1,2,3. Προσοχή σε αυτήν την περίπτωση: όταν τελειώσει η επανάληψη, η μεταβλητή games έχει την τιμή 3 και όχι την τιμή 4, όπως συμβαίνει στην περίπτωση της while.

Αν θέλαμε να εμφανίσουμε όλους τους θετικούς διψήφιους αριθμούς θα γράφαμε:

```
for number in range(10, 100) :
    print( number )
```

ενώ αν θέλαμε όλα τα θετικά πολλαπλάσια του 3 μικρότερα του 100, θα γράφαμε:

```
for number in range(3, 100, 3) :
    print( number )
```

όπου ο πρώτος αριθμός είναι ο αριθμός εκκίνησης, ο δεύτερος αριθμός είναι το πάνω όριο, το οποίο όμως δε φτάνουμε, και ο τρίτος αριθμός είναι το βήμα, δηλαδή η απόσταση μεταξύ δυο διαδοχικών αριθμών.

Η εντολή for games in range(4) εκτελεί τις εντολές που βρίσκονται μέσα της 4 φορές. Χρησιμοποιείται συνήθως όταν γνωρίζουμε εξαρχής πόσες επαναλήψεις θέλουμε να εκτελεστούν.

Παρακάτω δίνονται μερικά παραδείγματα χρήσης της συνάρτησης range.

range(5) = [0,1,2,3,4]

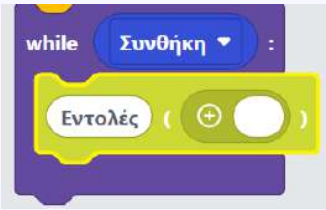
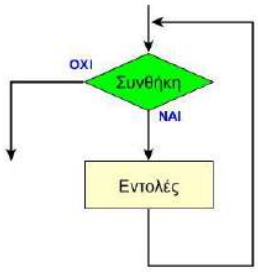
range(2,6) = [2,3,4,5]

range(1,10,2) = [1,3,5,7,9]

range(2,100,20) = [2,22,42]

```
import random
RPS = ["ΠΕΤΡΑ", "ΨΑΛΙΔΙ", "ΧΑΡΤΙ"]
wins = losses = 0
for games in range(4) :
    choice = random.randint(0, 2)
    computer = RPS[choice]
    player = input("Διάλεξε ΠΕΤΡΑ, ΨΑΛΙΔΙ ή ΧΑΡΤΙ : ")
    print(computer)
    if ( player=="ΠΕΤΡΑ"  and computer=="ΨΑΛΙΔΙ" or
         player=="ΨΑΛΙΔΙ" and computer=="ΧΑΡΤΙ"  or
         player=="ΧΑΡΤΙ"  and computer=="ΠΕΤΡΑ" ) :
      print("ΚΕΡΔΙΣΕΣ ΜΠΡΑΒΟ")
      wins += 1
    elif player==computer:
      print("ΙΣΟΠΑΛΙΑ")
    else:
      print("ΣΕ ΚΕΡΔΙΣΑ")
      losses+=1
print(wins, " - ", losses)
```

Η εντολή επανάληψης for είναι πολύ χρήσιμη, όμως η εντολή while είναι πιο γενική και μπορεί να χρησιμοποιηθεί σε όλες τις περιπτώσεις.

Δομή Επανάληψης while			
Όσο ισχύει η Συνθήκη Εντολές Τέλος_Επανάληψης		**while** <Συνθήκη>: Εντολές	

Προσοχή στις στοιχίσεις στη γλώσσα Python

Σε άλλες γλώσσες προγραμματισμού υπάρχουν δεσμευμένες λέξεις που οριοθετούν τις εντολές μέσα σε μια δομή επιλογής ή επανάληψης. Στην Python δεν υπάρχουν τέτοιες λέξεις-ενδείξεις. Οι εντολές που βρίσκονται μέσα σε μια δομή επανάληψης γράφονται μια εσοχή πιο δεξιά. Για παράδειγμα η εντολή Polygon(20, 8) θα εκτελεστεί 10 φορές, ενώ η εντολή Polygon(100, 6) μόνο μία.

```
for i in range(10):
  turtle.forward(20)
  Polygon(20, 8)
Polygon(100, 6)
```