Εφαρμογές Η/Υ

2ο Μάθημα

- Εργαλεία ελέγχου ροής
  - Εντολές `if`, `for`, `while`
  - Συναρτήσεις
- Σύνθετοι τύποι αντικειμένων
  - Πλειάδες
  - Λίστες
  - Σύνολα
  - Λεξικά

Ν. Λαγαρός, Θ. Στάμος, Χ. Φραγκουδάκης

Εισαγωγή στην Python

Εργαλεία ελέγχου ροής : Εντολή `if`

```
>>> x = int(input("Please enter an integer: "))
Please enter an integer: 42
>>> if x < 0:
...     x = 0
...     print('Negative changed to zero')
... elif x == 0:
...     print('Zero')
... elif x == 1:
...     print('Single')
... else:
...     print('More')
...
More
```

Εισαγωγή στην Python

Εργαλεία ελέγχου ροής: Εντολή `if`

```
>>> alist = [1, 2, 3, 4]
>>> if alist:                          # Ισοδύναμο με alist != []
...     print("Not Empty")
... else:                              # alist == []
...     print("Empty")
...
Not Empty
>>> anumber = 0
>>> if anumber:                        # Ισοδύναμο με anumber != 0
...     print("Not Zero")
... else:                              # anumber == 0
...     print("Zero")
...
Zero
>>> if not anumber:                    # Ισοδύναμο με anumber == 0
...     print("Zero")
...
Zero
```

- Τα διάφορα "κενά" αντικείμενα (π.χ. `0`, `''`, `[]`) αντιστοιχούν στο `False`
- Τα "μη κενά" αντικείμενα αντιστοιχούν στο `True`

Εισαγωγή στην Python

Εργαλεία ελέγχου ροής : Εντολή `if`

Η έκφραση ελέγχων με χρήση του `if` συντομεύεται μέσω της οκνηρής αποτίμησης

```
>>> answer = ''
>>> if answer == '':
...     print("Empty answer")
... else:
...     print(answer)
...
Empty answer
>>> print(answer or "Empty answer") # Ακριβώς ισοδύναμο με όλο το παραπάνω if
Empty answer
```

Εισαγωγή στην Python

Εργαλεία ελέγχου ροής : Εντολή `for`

```
>>> # Measure some strings:
... words = ['cat', 'window', 'defenestrate']
>>> for w in words:
...     print(w, len(w))
...
cat 3
window 6
defenestrate 12
```

- Επανάληψη με βάση μια ακολουθία αντικειμένων
- Τηρείται η διάταξη της ακολουθίας

Εισαγωγή στην Python

Εργαλεία ελέγχου ροής : Εντολή `for`

```
>>> for i in range(5):
...     print(i)
...
0
1
2
3
4
```

- Η συνάρτηση `range` παράγει μια ακολουθία διαδοχικών αριθμών
- Το όρισμα της `range` δεν ανήκει στη ακολουθία που παράγεται
- Μπορεί να οριστεί αρχική τιμή και βήμα:
  - `range(5, 9)` παράγει `5 6 7 8`
  - `range(0, 10, 3)` παράγει `0 3 6 9`
  - `range(-10, -100, -30)` παράγει `-10 -40 -70`

Εισαγωγή στην Python

Εργαλεία ελέγχου ροής: Εντολή `for`

```
>>> for n in range(2, 10):    # Εκτέλεση 8 επαναλήψεων
...     for x in range(2, n): # Εκτέλεση n-1 επαναλήψεων
...         if n % x == 0:    # Το x διαιρεί το n
...             print(n, 'είναι ίσο μέ', x, '*', n//x)
...             break # Πρώιμος τερματιμός των επαναλήψεων
...     else: # Εκτελείται όταν τελειώσουν κανονικά (όχι πρώιμα) οι επαναλήψεις
...         # Δεν βρέθηκε κάποιος παράγοντας του n
...         print(n, 'είναι πρώτος αριθμός')
...
2 είναι πρώτος αριθμός
3 είναι πρώτος αριθμός
4 είναι ίσο μέ 2 * 2
5 είναι πρώτος αριθμός
6 είναι ίσο μέ 2 * 3
7 είναι πρώτος αριθμός
8 είναι ίσο μέ 2 * 4
9 είναι ίσο μέ 3 * 3
```

- Η εντολή `break` τερματίζει πρώιμα την επανάληψη
- Η `for` μπορεί να έχει `else` που εκτελείται όταν οι επαναλήψεις τελειώσουν κανονικά (όχι πρώιμα)

Εισαγωγή στην Python

Εργαλεία ελέγχου ροής : Εντολή `for`

```
>>> for letter in 'Python':
...     if letter == 'h':
...             continue
...     print('Τρέχον γράμμα:', letter)
... else:
...     print('Τέλος')
...
Τρέχον γράμμα: P
Τρέχον γράμμα: y
Τρέχον γράμμα: t
Τρέχον γράμμα: o
Τρέχον γράμμα: n
Τέλος
```

- Η εντολή `continue` απορρίπτει όλες τις υπόλοιπες εντολές της τρέχουσας επανάληψης και προωθεί τον έλεγχο στην αρχή της επόμενης επανάληψης

Εισαγωγή στην Python

Εργαλεία ελέγχου ροής : Εντολή `while`

```
>>> # Ακολουθία Fibonacci
... # το άθροισμα δύο στοιχείων ορίζει το επόμενο
... a, b = 0, 1
>>> while b < 10:
...     print(b)
...     a, b = b, a+b
... else: # Εκτελείται όταν η συνθήκη του while γίνει False
...     print("Τέλος της ακολουθίας")
...
1
1
2
3
5
8
Τέλος της ακολουθίας
```

- Οι εντολές εκτελούνται όσο η έκφραση `b < 10` έχει την τιμή `True`
- Η `while` μπορεί να έχει `else` που εκτελείται όταν η συνθήκη της επανάληψης γίνει `False`

Εισαγωγή στην Python

Εργαλεία ελέγχου ροής : Συναρτήσεις

```
>>> def fib(n):    # εκτυπώνει την ακολουθία Fibonacci έως τον αριθμό n
...     """Prints the Fibonacci series up to n."""
...     a, b = 0, 1
...     while a < n:
...         print(a, end=' ')
...         a, b = b, a+b
...     print()
...
>>> # Τώρα καλούμε τη συνάρτηση που μόλις ορίσαμε:
... fib(2000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
>>> f = fib
>>> f(100)
0 1 1 2 3 5 8 13 21 34 55 89
```

- Ορίζονται με χρήση του `def`, ένα όνομα και προεραιτικές παραμέτρους
- Τεκμηριώνονται με ένα string αμέσως μετά τον ορισμό τους
- Ο ορισμός μιας συνάρτησης εισάγει ένα νέο όνομα όπως και στις μεταβλητές

Εισαγωγή στην Python

Εργαλεία ελέγχου ροής : Συναρτήσεις

```
>>> x = 1
>>> id(x)
10911712
>>> def change(x):
...     x = x + 1
...     print(x, id(x))
...
>>> change(x)
2 10911744
>>> x
1
>>> id(x)
10911712
```

- Οι παράμετροι γίνονται τοπικές μεταβλητές στο σώμα της συνάρτησης

Εισαγωγή στην Python

Εργαλεία ελέγχου ροής : Συναρτήσεις

```
>>> def fib2(n):   # επιστρέφει την ακολουθία Fibonacci έως τον αριθμό n
...     """Returns a list containing the Fibonacci series up to n."""
...     result = []
...     a, b = 0, 1
...     while a < n:
...         result.append(a)    # Προσθέτει το a στο τέλος της λίστας result
...         a, b = b, a+b
...     return result
...
>>> f100 = fib2(100)     # Κλήση της συνάρτησης
>>> f100                 # Εκτύπωση της λίστας που επιστρέφει η fib2
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

- Η εντολή `return` επιστρέφει τα ορίσματά της στο κυρίως πρόγραμμα
- Αν δεν υπάρχει `return` επιστρέφει το ειδικό αντικείμενο `None`

Εισαγωγή στην Python

Εργαλεία ελέγχου ροής : Συναρτήσεις

```
>>> def f(x):
...     return x**2
...
>>> f(8)
64
>>> g = lambda x: x**2
>>> g(8)
64
>>> h = lambda x,y: x**y
>>> h(4,2)
16
```

- Οι ανώνυμες συναρτήσεις ορίζονται με χρήση του `lambda`
- Οι συναρτήσεις `lambda` επιστρέφουν τον υπολογισμό μιας έκφρασης
- Μπορούν να χρησιμοποιηθούν όπου μπορεί να χρησιμοποιηθεί μια συνάρτηση

Εισαγωγή στην Python

Εργαλεία ελέγχου ροής : Συναρτήσεις

```
>>> def make_incrementor(n):
...     return lambda x: x + n
...
>>> f = make_incrementor(2)
>>> g = make_incrementor(6)
>>> print(f(42), g(42))
44 48
>>> print(make_incrementor(22)(33))
55
```

Παράδειγμα χρήσης ανώνυμης συνάρτησης

- Η `make_incrementor` ορίζει μια ανώνυμη συνάρτηση και την επιστρέφει
- Η ανώνυμη συνάρτηση προσθέτει στο όρισμά της το `n`
- Ορίζουμε πολλές `make_incrementor` με διαφορετικά ορίσματα
- Η συνάρτηση `f` είναι ανεξάρτητη της `g`
- Δεν είναι αναγκαίο να ανατεθεί σε μεταβλητή για να χρησιμοποιηθεί

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων

Στην Python υπάρχουν σύνθετοι τύποι αντικειμένων που χρησιμοποιούνται για την ομαδοποίηση άλλων αντικειμένων.

- Λίστες (lists)
- Πλειάδες (tuples)
- Σύνολα (sets)
- Λεξικά (dictionaries)

```
>>> alist = ['spam', 'eggs', 100, 100.321, [1, 2]]
>>> atuple = ('spam', 'eggs', 100, 100.321, [1, 2])
>>> aset = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
>>> adict = {'jack': 4098, 'mary': 3422, 'guido': 4127}
```

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων : Λίστες

- Περιέχουν άλλα αντικείμενα, χωρισμένα με κόμμα, ανάμεσα σε `[` και `]`
- Τα αντικείμενα μπορεί να είναι διάφορων τύπων
- Η δεικτοδότηση των περιεχομένων μιας λίστας αρχίζει με τον αριθμό `0`
- Η συνάρτηση `len` επιστρέφει το μήκος της λίστας

```
>>> alist = ['spam', 'eggs', 100, 100.321, [1, 2]]
>>> alist[0]
'spam'
>>> alist[3]
100.321
>>> alist[-1]
[1, 2]
>>> len(alist)
5
>>> len(alist[-1])
2
```

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων : Λίστες

```
>>> a = [1, 2, 3]
>>> b = a
>>> id(a)
139691426785480
>>> id(b)
139691426785480
>>> b[2] = 'changed'
>>> b
[1, 2, 'changed']
>>> id(b)
139691426785480
>>> a
[1, 2, 'changed']
>>> id(a)
139691426785480
```

- Οι λίστες είναι αντικείμενα που μεταλλάσσονται (mutable objects)

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων : Λίστες

```
  -9  -8  -7  -6  -5  -4  -3  -2  -1
   0   1   2   3   4   5   6   7   8
 +---+---+---+---+---+---+---+---+---+
 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
 +---+---+---+---+---+---+---+---+---+
 0   1   2   3   4   5   6   7   8   9
-9  -8  -7  -6  -5  -4  -3  -2  -1
```

Δείκτες θέσεων της λίστας

alist = [1, 2, 3, 4, 5, 6, 7, 8, 9]

Δείκτες ορίων για τον τεμαχισμό

```
>>> alist[2:5]
[3, 4, 5]
>>> alist[-7:-4]
[3, 4, 5]
>>> alist[1:-1]
[2, 3, 4, 5, 6, 7, 8]
```

- Χρησιμοποιούνται θετικοί και αρνητικοί δείκτες
  - `alist[3]` είναι το 4ο στοιχείο (το `4`)
  - `alist[-2]` είναι το 2ο στοιχείο από το τέλος (το `8`)
- Το τεμάχιο (slice) της λίστας από το `i` έως το `j` περιέχει όλα τα αντικείμενα μεταξύ των ορίων με δείκτες `i` και `j`

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων : Λίστες

```
  +---+---+---+---+---+---+---+---+---+
  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |      a = [1, 2, 3, 4, 5, 6, 7, 8, 9]
  +---+---+---+---+---+---+---+---+---+
  0   1   2   3   4   5   6   7   8   9      Δείκτες ορίων για τον τεμαχισμό
```

```
>>> a[:6]                                  # Από την αρχή έως το 6ο όριο
[1, 2, 3, 4, 5, 6]
>>> a[8:]                                  # Από το 8ο όριο έως το τέλος
[9]
>>> a[:]                                   # Από την αρχή έως το τέλος (αντίγραφο)
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> a[0:3] = [11, 22, 33]                  # Αντικατάσταση αντικειμένων
>>> a
[11, 22, 33, 4, 5, 6, 7, 8, 9]
>>> a[0:3] = []                            # Διαγραφή αντικειμένων
>>> a
[4, 5, 6, 7, 8, 9]
>>> a[:0] = [1, 2, 3]                      # Εισαγωγή αντικειμένων στην αρχή
>>> a
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> a[5:5] = ['slipped', 'into']           # Εισαγωγή αντικειμένων στο 5ο όριο
>>> a
[1, 2, 3, 4, 5, 'slipped', 'into', 6, 7, 8, 9]
```

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων : Λίστες

- Οι μέθοδοι είναι διαδικασίες που επιδρούν στα δεδομένα του αντικειμένου
- Οι μέθοδοι μπορεί να λαμβάνουν σαν παραμέτρους άλλα αντικείμενα
- Χρησιμοποιούμε μια μέθοδο βάζοντας ένα `.` μετά το αντικείμενο, το όνομα της μεθόδου και τις παραμέτρους μέσα σε παρενθέσεις:

```
alist.insert(3, [1, 'a'])
```

- Η λίστα στην Python είναι εφοδιασμένη με διάφορες μεθόδους (τα `an_object`, `a_list`, `position` είναι ονόματα αντικειμένων):
  - `append(an_object)`
  - `extend(a_list)`
  - `insert(position, an_object)`
  - `remove(an_object)`
  - `pop( [ position ] )` (τα [] υποδηλώνουν προαιρετική παράμετρο)
  - `index(an_object)`
  - `count(an_object)`
  - `sort()`
  - `reverse()`

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων : Λίστες

```
>>> a = [66.25, 333, 333, 1, 1234.5]
>>> print(a.count(333), a.count(66.25), a.count('find_x'))
2 1 0
>>> a.insert(2, -1)                    # Εισαγωγή του -1 στο 2ο όριο
>>> a
[66.25, 333, -1, 333, 1, 1234.5]
>>> a.append(333)                      # Προσθήκη του 333 στο τέλος της λίστας
>>> a
[66.25, 333, -1, 333, 1, 1234.5, 333]
>>> a.index(333)                       # Σε ποιο δείκτη βρίσκεται το 333
1
>>> a.remove(333)                      # Εξαγωγή του 333 από τη λίστα
>>> a
[66.25, -1, 333, 1, 1234.5, 333]
>>> a.reverse()                        # Αλλαγή επιτόπου στην αντίστροφη λίστα
>>> a
[333, 1234.5, 1, 333, -1, 66.25]
>>> a.sort()                           # Αλλαγή επιτόπου σε ταξινομημένη λίστα
>>> a
[-1, 1, 66.25, 333, 333, 1234.5]
>>> x, y = a.pop(0), a.pop()           # Εξαγωγή από την 1η θέση και την τελευταία
>>> x, y
(-1, 1234.5)
>>> a
[1, 66.25, 333, 333]
```

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων : Λίστες

```
Χρήση της λίστας σαν στοίβα       +---+
                                  | d |
                                  +---+
                                  | c |
                                  +---+
                                  | b |
                                  +---+
(εισαγωγή στη στοίβα) a b c d ... | a | ... d c b a (εξαγωγή από τη στοίβα)
                                  +---+
```

```
>>> stack = []
>>> stack.append('a'); stack.append('b'); stack.append('c'); stack.append('d')
>>> stack
['a', 'b', 'c', 'd']
>>> stack.pop(); stack.pop(); stack.pop(); stack.pop()
'd'
'c'
'b'
'a'
>>> stack
[]
```

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων : Λίστες

Πρόβλημα: Είναι μια σειρά παρενθέσεων "ζυγισμένη";

```
Η πιο πρόσφατα ανοικτή παρένθεση κλείνει με την πρώτη κλειστή
         ^ ^
         | |
( ( ) ( ( ) ) ( ) )
|                 |
v                 v
Η πρώτη ανοικτή παρένθεση μπορεί να πρέπει να περιμένει την τελευταία για κλείσει
```

- Ξεκινάμε με άδεια στοίβα και όταν συναντάμε `(` τη στοιβάζουμε (`append`)
- Όταν συναντάμε `)` βγάζουμε μια `(` από την κορυφή της στοίβας (`pop`)
- Όσο είναι δυνατό να γίνεται `pop` οι παρανθέσεις παραμένουν ζυγισμένες
- Αν συναντήσουμε `)` και η στοίβα είναι άδεια η σειρά δεν είναι ζυγισμένη
- Όταν τελειώσουν οι παρενθέσεις εξετάζουμε αν η στοίβα είναι άδεια
- Αν η στοίβα είναι άδεια η σειρά των παρενθέσεων είναι ζυγισμένη

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων : Λίστες

```
>>> def balancedParenthesis(symbols):
...     stack = []
...     balanced = True
...     for symbol in symbols:          # Για κάθε σύμβολο από αριστερά προς δεξιά
...         if symbol == '(':
...             stack.append(symbol)    # Στοιβάζει την ανοικτή παρένθεση
...         else:                       # Βρέθηκε κλειστή παρένθεση
...             if not stack:           # Αν η στοίβα είναι άδεια
...                 balanced = False    # Η σειρά δεν είναι ζυγισμένη
...                 break
...             else:
...                 stack.pop()         # Εξαγωγή παρένθεσης από τη στοίβα
...     return balanced and not stack
...
>>> balancedParenthesis('(()()()())')
True
>>> balancedParenthesis('((((((((((()))))))))))')
True
>>> balancedParenthesis('((()(()))(()))')
True
>>> balancedParenthesis('(())((()))')
False
```

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων : Λίστες

```
>>> squares = []
>>> for x in range(10):       # Για κάθε αριθμό μέχρι και το 9
...     squares.append(x**2)
...
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> squares = [x**2 for x in range(10)]
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

- Μια λίστα δημιουργείται συνοπτικά με την περιγραφή της (list comprehension)
- Μια έκφραση επιδρά σε κάθε αντικείμενο μιας ομαδοποίησης αντικειμένων
- Οι περιγραφές περικλείονται σε `[` και `]` και ορίζουν τοπικά ονόματα
- Οι περιγραφές περιέχουν τουλάχιστον ένα `for`

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων : Λίστες

```
>>> vec = [-4, -2, 0, 2, 4]
>>> [x*2 for x in vec]          # Νέα λίστα με τις διπλάσιες τιμές του vec
[-8, -4, 0, 4, 8]

>>> [x for x in vec if x >= 0] # Νέα λίστα με τις θετικές τιμές του vec
[0, 2, 4]

>>> [abs(x) for x in vec]       # Νέα λίστα με συνάρτηση τιμών του vec
[4, 2, 0, 2, 4]
```

```
>>> vec = [[1,2,3], [4,5,6], [7,8,9]]
>>> [num for elem in vec for num in elem]
[1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> from math import pi
>>> [str(round(pi, i)) for i in range(5)]
['3.1', '3.14', '3.142', '3.1416', '3.14159']
```

- Σε μια περιγραφή λίστας μια έκφραση μπορεί να επιδρά υπό συνθήκη
- Μπορεί να επιδρούν οσοδήποτε πολύπλοκες εκφράσεις
- Μπορεί να υπάρχουν πολλαπλά `for`

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων : Λίστες

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}^T = \begin{bmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{bmatrix}$$

```
>>> matrix = [
...     [1,  2,  3,  4],
...     [5,  6,  7,  8],
...     [9, 10, 11, 12]
... ]
>>> [[row[i] for row in matrix] for i in range(4)]
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

- Η αρχική έκφραση μιας περιγραφής λίστας μπορεί να είναι περιγραφή λίστας

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων : Λίστες

```
>>> matrix = [
...     [1,  2,  3,  4],
...     [5,  6,  7,  8],
...     [9, 10, 11, 12]
... ]
>>> [[row[i] for row in matrix] for i in range(4)]
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]

>>> transposed = []
>>> for i in range(4):
...     transposed.append([row[i] for row in matrix])
>>> transposed
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]

>>> transposed = []
>>> for i in range(4):
...     transposed_row = []
...     for row in matrix:
...             transposed_row.append(row[i])
...     transposed.append(transposed_row)
>>> transposed
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων : Λίστες

```
>>> alist = [1, 2, 3, 4]
>>> def change(alist):
...     alist[3] = 'four'
...     print(alist)
...
>>> change(alist)
[1, 2, 3, 'four']
>>> alist
[1, 2, 3, 'four']
```

- Οι λίστες μπορούν να μεταλλάσσονται όταν περνούν σαν ορίσματα σε συναρτήσεις

Ερωτήσεις;