Ψηφιακή Επεξεργασία Εικόνων
ΗΥ-371

Φροντιστήριο 1: Εισαγωγή στην Python
03 / 10 / 2023

Βοηθός: Ιωάννης Καζιάλες ~ kaziales@csd.uoc.gr

Λίγα λόγια για την Python

Python www.python.org

- open-source, high-level, interpreted γλώσσα προγραμματισμού γενικού σκοπού
- έχει δυναμικό σύστημα τύπων και αυτόματη διαχείριση μνήμης
- απλή σύνταξη παρόμοια με την αγγλική γλώσσα, χρειάζεται λιγότερες γραμμές κώδικα
- easy-to-learn, easy-to-read, easy-to-maintain
- υπάρχουν διαθέσιμες πολλές χρήσιμες βιβλιοθήκες

ΣΗΜΑΝΤΙΚΟ: Οι εκδόσεις 2.x και 3.x της python έχουν αρκετές διαφορές! Στο μάθημα: python >= 3.6

Χρήσιμοι Σύνδεσμοι:

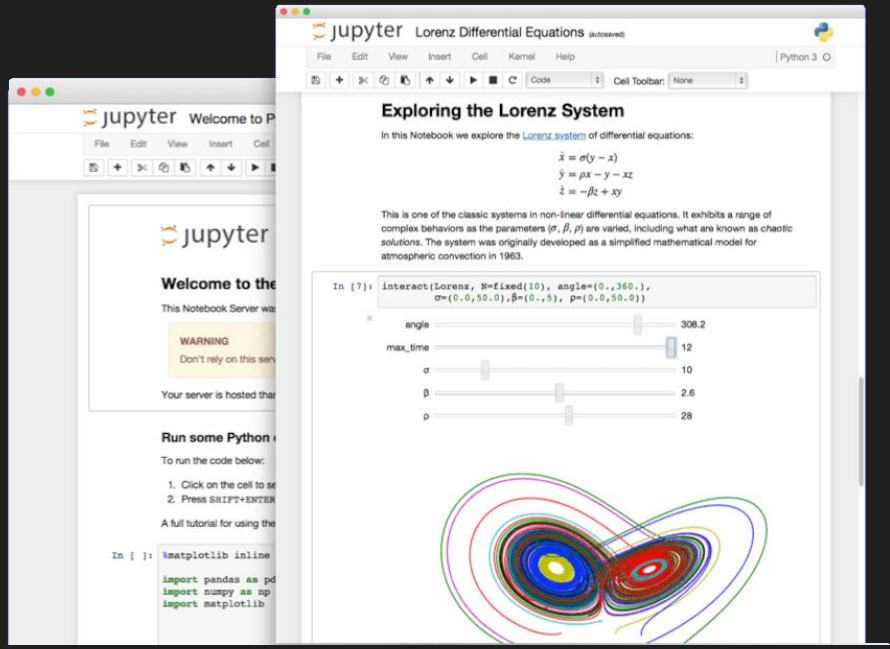
- official tutorial: docs.python.org/3/tutorial/
- google's python class: developers.google.com/edu/python
- W3Schools tutorial: www.w3schools.com/python/
- Geeks for Geeks tutorial: www.geeksforgeeks.org/python-programming-language/
- tutorialspoint tutorial: www.tutorialspoint.com/python/index.htm

Εγκατάσταση/Ρύθμιση του environment σας

- Δύο επιλογές
  - Μπορείτε να **στήσετε το environment σας χειροκίνητα**, εγκαθιστώντας python kernels και packages χειροκίνητα (πιθανά σφάλματα, conflicts αργότερα)
  - Μπορείτε να **χρησιμοποιήσετε το Anaconda** (highly recommended από εμάς)
- Το Anaconda είναι μια πλατφόρμα για Linux, Windows, και Mac OS που βοηθάει στη χρήση Python/R για data science και machine learning
- Βοηθάει στην εύκολη δημιουργία virtual environments και την εγκατάσταση πακέτων, ελαχιστοποιώντας τα λάθη
- έχει προεγκατεστημένα IDEs για python (Spyder) και χρήσιμες εφαρμογές
- Εγκατάσταση Anaconda: οδηγός video
- Conda Cheatsheet

Jupyter (IPython) Notebook

- Το Jupyter Notebook είναι μια web εφαρμογή που επιτρέπει να δημιουργήσεις και να μοιραστείς αρχεία (.ipynb) που περιέχουν κώδικα, εξισώσεις, visualizations και επεξηγηματικό κείμενο. Τρέχει locally στον browser.
- Το Google Colab ουσιαστικά είναι notebook που τρέχει εξ ολοκλήρου στο cloud. Δεν χρειάζεται εγκατάσταση και σου παρέχει δωρεάν υπολογιστικούς πόρους (CPUs, GPUs και TPUs).
- Jupyter Notebook Tutorial
- σύντομο tutorial markdown



Βασικό Συντακτικό Python

- Βασικοί Τύποι
- Σύνθετοι Τύποι
- Βασικοί Τελεστές
- Πράξεις σε ακολουθίες
- Μέθοδοι για Λίστες
- Είσοδος/Έξοδος
- Εκτέλεση υπό συνθήκη
- Επαναλήψεις
- list comprehension
- Συναρτήσεις

Βασικοί Τύποι

Αριθμοί

- **int** - ακέραιοι
- **float** - αριθμοί κινητής υποδιαστολής

Λογικές τιμές - **bool** (True/False)

Συμβολοσειρές (strings) - **str**

```
int("1305")      # 1305
float("3.14")    # 3.14
int(39.99)       # 39
str(371)         # '371'
str(-3.14)       # '-3.14'
```

```
# ints
int1 = 136
int2: int = -8

# floats
float1 = 3.1415
float2: float = -1.89
float3 = 2.48e4 # = 2.48 * 10^4 = 24800

# bools
bool1 = True
bool2: bool = False

# strs
str1 = "Με διπλά εισαγωγικά"
str2: str = 'Με μονά εισαγωγικά'
str3 = "\"φράση\" με escape character"
str4 = "'φράση' με άλλα εισαγωγικά"
str5 = "ειδικοί \ηχαρακτήρες"
```

Σύνθετοι Τύποι (1/2)

list - λίστα

- ordered (διατεταγμένη)
- mutable (μεταλλάξιμη)
- ορίζεται με [ ]
- συνήθως τιμές ίδιου τύπου
- όχι ακριβώς πίνακας

tuple - πλειάδα

- ordered (διατεταγμένη)
- immutable (αμετάβλητη)
- ορίζεται με ( )

```
1 # lists
2 L1 = [1, 2, 3]
3 L2 = [1, "string", True]
4 L3 = [L1, L2]
5 L4 = [] # άδεια λίστα
6
7 # tuples
8 t1 = (1, 2, 3)
9 t2 = (1, "string", True)
10 t4 = () # άδειο tuple
11 t3 = ("ένα αντικείμενο",) # χρειάζεται ,
12
13 # Προσοχή
14 (3.14,) # tuple με ένα στοιχείο
15 (3.14)  # χωρίς , είναι ένα float
```

Σύνθετοι Τύποι (2/2)

set - σύνολο

- unordered (χωρίς διάταξη)
- mutable (μεταλλάξιμη)
- ορίζεται με { }
- εκτελεί πράξεις συνόλων (τομή, ένωση, διαφορά, κλπ)
- κάθε στοιχείο 1 φορά μόνο

dict - λεξικό

- αντιστοιχεί κλειδιά σε τιμές
- mutable (μεταλλάξιμη)
- ορίζεται ως {κλειδί : τιμή}
- ουσιαστικά hash table/map

```python
# sets
s1 = {1, 2, 2, 3} # {1, 2, 3}
s2 = {1, "string", True}
s3 = set()  # άδειο σύνολο

# dicts
d1 = {"όνομα":"Γιάννης", "ηλικία":23}
d2 = {1:"ένα", 2:"δύο"}
d3 = {t1:12, True:"str", 1: False}
d4 = {} # άδειο dict
```

Βασικοί Τελεστές (1/2)

Αριθμητικοί Τελεστές

Τελεστής	Περιγραφή
`x + y`	πρόσθεση
`x - y`	αφαίρεση
`x * y`	πολλαπλασιασμός
`x / y`	πραγματική διαίρεση
`x // y`	ακέραια διαίρεση
`x % y`	υπόλοιπο ακέραιας διαίρεσης
`x ** y`	ύψωση σε δύναμη

Τελεστές Εκχώρησης

Τελεστής	Περιγραφή
`x = y`	εκχώρηση
`x += y`	x = x + y
`x -= y`	x = x -y
`x *= y`	x = x * y
`x /= y`	x = x / y
`x %= y`	x = x % y

```
1 a = 12
2 b = 34.56
3 c = a + b          # c = 46.56
4 d = a - b          # d = -22.56
5 e = (a + b) * 2    # e = 93.12
6 f = a / 10         # f = 1.2
7 g = a // 10        # g = 1
8 h = a % 10         # h = 2
9 a += 5             # a = 17
10 a -= 4            # a = 13
11 a *= 2            # a = 26
```

Βασικοί Τελεστές (2/2)

Τελεστές για strings

Τελεστής	Περιγραφή
`x + y`	συνένωση των strings
`x += y`	συνένωση των strings

Λογικοί Τελεστές

Τελεστής	Περιγραφή
`x and y`	λογική σύζευξη
`x or y`	λογική διάζευξη
`not x`	λογική άρνηση

Συσχετιστικοί Τελεστές

Τελεστής	Περιγραφή
`x == y`	ισότητα
`x !=y`	ανισότητα
`x > y`	μεγαλύτερο
`x < y`	μικρότερο
`x >= y`	μεγαλύτερο ή ίσο
`x <= y`	μικρότερο ή ίσο

```python
s = "Hello"
s2 = s + "371"        # s2 = "Hello371"
s += " world"         # s = "Hello world"

a, b = 18, 34.56
b1 = a == 18          # b1 = True
b2 = b > 34           # b2 = True
b3 = s == "Hi"        # b3 = False
b4 = s > "Alpha"      # b4 = True
b5 = not b3           # b5 = True
b6 = b1 and b2        # b6 = True
b7 = b1 or b3         # b7 = True
b8 = b1 and b3        # b8 = False
b9 = a<10 and b7      # b9 = False
```

Πράξεις σε ακολουθίες (lists, tuples, strings) (1/2)

Τελεστής/Συνάρτηση	Περιγραφή
`seq[index]`	Επιστρέφει το στοιχείο της ακολουθίας στη θέση index. Το index ξεκινάει από 0
`seq[start:stop:step]`	slicing: Επιστρέφει μια υποακολουθία της αρχικής στο διάστημα [start, stop). Επιλέγονται τα στοιχεία : start, start + step, start + 2step, ...
`len(seq)`	Επιστρέφει το μήκος της ακολουθίας (πλήθος στοιχείων)
`seq1 + seq2`	Δημιουργεί μια νέα ακολουθία που είναι συνένωση των δύο ακολουθιών
`n * seq`	Δημιουργεί μια νέα ακολουθία όπου η ακολουθία seq επαναλαμβάνεται n φορές
`e in seq`	Ελέγχει αν το στοιχείο e περιέχεται στην ακολουθία seq
`e not in seq`	Ελέγχει αν το στοιχείο e δεν περιέχεται στην ακολουθία seq

```python
L = [0, 1, 2, 3]

# indexing
x  = L[2] + L[-1]    # x = 2 + 3 = 5

# slicing
L1 = L[1:3]      # L1=[1, 2]
L2 = L[1:]       # L2=[1, 2, 3]
L3 = L[:]        # L3=[0, 1, 2, 3]
L4 = L[1::2]     # L4=[1, 3]
L5 = L[::-1]     # L5=[3, 2, 1, 0]

# μήκος
l = len(L)       # l = 4

# συνένωση
L6 = L1 + L4 # L6=[1, 2]+[1, 3]=[1, 2, 1, 3]
L7 = 2 * L1  # L7=2*[1, 2]=[1, 2, 1, 2]

# υπάρχει ένα στοιχείο στη λίστα?
b1 = 1 in L         # b1 = True
b2 = 3 not in L     # b2 = False
```

Πράξεις σε ακολουθίες (lists, tuples, strings) (2/2)

Τελεστής/Συνάρτηση	Περιγραφή
`seq.index(val)`	Επιστρέφει τη θέση της πρώτης εμφάνισης της τιμής val στην ακολουθία seq
`seq.count(val)`	Επιστρέφει το πλήθος των εμφανίσεων της τιμής val στην ακολουθία seq
`sorted(seq,[reverse])`	Επιστρέφει μια λίστα με τα στοιχεία της ακολουθίας seq ταξινομημένα. Βάζοντας reverse = True, η ταξινόμηση γίνεται σε φθίνουσα διάταξη
`reversed(seq)`	Επιστρέφει μια **νέα ακολουθία** με τα στοιχεία της seq αντεστραμμένα
`list(seq)`	Επιστρέφει μια **νέα λίστα** με τα στοιχεία της ακολουθίας seq

```python
L  = [0, 1, 2, 3]
L6 = [1, 2, 1, 3]

L8 = list(reversed(L))  # L8=[3, 2, 1, 0]
L9 = sorted(L7)         # L9=[1, 1, 2, 2]

c = L6.count(1)         # c = 2
d = L.index(3)          # d = 3
```

```python
s = "Programming"
s1 = s[2] + s[-1]    # s1 = "og"
s2 = s[1:9:3]        # s2 = "rrm"
s3 = 3 * "la"        # s3 = "lalala"
b3 = "a" in s        # b3 = True
w = sorted("help")   # w = ["e","h","l","p"]
x = sorted("help", reverse = True)
```

Μέθοδοι για λίστες

Τελεστής/Συνάρτηση	Περιγραφή
`L.append(val)`	Προσθέτει το στοιχείο val στο τέλος της λίστας L
`L.extend(seq)`	Προσθέτει όλα τα στοιχεία της ακολουθίας seq στο τέλος της λίστας L
`L.insert(index, val)`	Προσθέτει το στοιχείο val στη θέση *index* της λίστας L
`L.remove(val)`	Αφαιρεί από τη λίστα L το πρώτο στοιχείο με τιμή val
`L.pop()`	Αφαιρεί και επιστρέφει το τελευταίο στοιχείο της λίστας L
`L.sort()`	Ταξινομεί τη λίστα L in-place (δεν επιστρέφει κάτι)
`L.reverse()`	Αντιστρέφει τη λίστα L in-place (δεν επιστρέφει κάτι)

```
1 L = []
2 L.append(1)          # L=[1]
3 L.extend([2, 3, 2]) # L=[1, 2, 3, 2]
4 L.insert(1, 4)       # L=[1, 4, 2, 3, 2]
5 L.remove(2)          # L=[1, 4, 3, 2]
6 L.reverse()          # L=[2, 3, 4, 1]
7 L.sort()             # L=[1, 2, 3, 4]
```

Είσοδος / Έξοδος

input - είσοδος

- διαβάζει είσοδο του χρήστη
- ως όρισμα παίρνει ένα επεξηγηματικό κείμενο που τυπώνει πριν διαβάσει δεδομένα
- επιστρέφει string, οπότε ίσως χρειαστεί να γίνει μετατροπή τύπου

```python
name = input("Πως σε λένε;")
age = int(input("Τι ηλικία έχεις;"))

print("hello world")

# τα παρακάτω τυπώνουν το ίδιο κείμενο
print("Γειά σου " + name + ". Είσαι " + str(age) + " χρονών.")
print("Γειά σου", name, ". Είσαι", age, "χρονών.")
print("Γειά σου {}. Είσαι {} χρονών.".format(name, age))
# καλύτερη μέθοδος : f-strings (python >= 3.6)
print(f"Γειά σου {name}. Είσαι {age} χρονών.")

# πχ, Γειά σου Γιάννη. Είσαι 23 χρονών.
```

print - έξοδος

- τυπώνει κείμενο

```python
x = 4863.4343091
print(x)            # 4863.4343091
print(f"{x:.2f}")   # 4863.43 → 2 δεκαδικά ψηφία
```

Εκτέλεση υπό συνθήκη

if - elif - else

- υποχρεωτικά το if στην αρχή, προαιρετικά ακολουθείται από elif συνθήκες και στο τέλος else
- κάθε κομμάτι ακολουθείται από το “σώμα”, εντολές σε εσοχή που εκτελούνται αν η συνθήκη πέτυχε
- Εκτελείται το πρώτο σώμα (μόνο ένα) που έχει αληθή συνθήκη
- Αν όλες οι συνθήκες είναι ψευδείς, εκτελείται το σώμα του else (αν υπάρχει)

```
1 if x > 0:
2     print("το x είναι θετικό")
```

```
1 if grade >= 8.5:
2     description = "Άριστα"
3 elif grade >= 6.5:
4     description = "Λίαν Καλώς"
5 elif grade >= 5:
6     description = "Καλώς"
7 else:
8     description = "Ανεπιτυχώς"
```

Επαναλήψεις (1/3)

Επανάληψη **while**

- ελέγχεται πρώτα η συνθήκη. Αν είναι αληθής, μπαίνουμε στο σώμα της while, αλλιώς συνεχίζουμε
- στο σώμα εκτελούμε τις εντολές και μετά επιστρέφουμε πάλι στον έλεγχο της συνθήκης και επαναλαμβάνουμε
- μπορούμε να αλλάξουμε τη ροη του προγράμματος, χρησιμοποιώντας:
  - **continue**:
    συνεχίζει στην επόμενη επανάληψη
  - **break**:
    μας βγάζει έξω από την επανάληψη
- στο τέλος μπορεί να περιέχει ένα **else**, που εκτελείται αν η συνθήκη γίνει ψευδής
- χρησιμοποιείται συνήθως όταν έχουμε μια συνθήκη που αλλάζει με τις επαναλήψεις

```
1 s, i = 0, 0
2 while i < 10:
3     s = s + i
4     i = i + 1
5 # s = 0 + 1 + 2 + ...+ 9 = 45
```

```
1 i = 0
2 while i < 10:
3     if i == 3:
4         i += 1
5         continue
6     print(i)
7     if i == 5:
8         break
9     i += 1
10 # τυπώνει 0 1 2 4 5
```

Επαναλήψεις (2/3)

Επανάληψη **for**

- εκτελείται για όλα τα στοιχεία μιας ακολουθίας και μας δίνει το επόμενο στοιχείο της ακολουθίας κάθε φορά
- ακόμα κι αν αλλάξουμε τη μεταβλητή στο σώμα, στην επόμενη επανάληψη θα πάρει την τιμή του επόμενου στοιχείου
- μπορούμε να αλλάξουμε τη ροη του προγράμματος, χρησιμοποιώντας:
  - **continue**:
    συνεχίζει στην επόμενη επανάληψη
  - **break**:
    μας βγάζει έξω από την επανάληψη
- στο τέλος μπορεί να περιέχει ένα **else**, που εκτελείται αν η επανάληψη τελειώσει κανονικά (χωρίς *break*)

```python
numbers = [1, 2, 6, 5, 8, 31, 22]
for num in numbers:
    if num % 2 != 0:
        print(num)
# τυπώνει 1 5 31
```

```python
text, counter = "ενα κειμενο", 0
for char in text:
    if char == "ε":
        counter += 1
# counter = 3
```

```python
values = (1, 2, 3)
for val in values:
    print(val)
    val = 0
# τυπώνει 1 2 3
```

Επαναλήψεις (3/3)

range(*[start,]* stop *[,step]* **)**

- παράγει μια ακολουθία από αριθμούς
- ξεκινάει από το *start* (0), αυξάνεται με βήμα *step* (1) και τελειώνει πριν το stop (χωρίς το stop)
- χρήσιμο για συγκεκριμένο αριθμό επαναλήψεων

enumerate(ακολουθία **)**

- επιστρέφει το index και την τιμή κάθε στοιχείου μιας ακολουθίας

```
1 range(4)          # 0, 1, 2, 3
2 range(2, 6)       # 2, 3, 4, 5
3 range(2, 10, 3) # 2, 5, 8
4 range(4, 1, -1) # 4, 3, 2
5
6 numbers = [12, 48, 53]
7
8 for i in range(len(numbers)):
9     print(i, "->", numbers[i])
10
11 for i, num in enumerate(numbers):
12     print(i, "->", num)
```

List Comprehension

- Μας επιτρέπει να φτιάξουμε μια λίστα με βάση τα στοιχεία κάποια άλλης ακολουθίας
- Μπορούμε να φιλτράρουμε στοιχεία της αρχικής ακολουθίας με τη χρήση καθώς και να περιγράψουμε δεδομένα που προκύπτουν από nested loops
- μας γλιτώνει γραμμές κώδικα

```python
# τετράγωνα για τους αριθμούς στο [1,20]
# 1ος τρόπος: επανάληψη
squares = []
for i in range(1, 21):
    squares.append(i**2)

# 2ος τρόπος: list comprehension
squares=[i**2 for i in range(1,21)]
```

```python
# τετράγωνα για τους ζυγούς αριθμούς στο [1,20]
# 1ος τρόπος: επανάληψη
squares = []
for i in range(1, 21):
    if i % 2 == 0:
        squares.append(i**2)

# 2ος τρόπος: list comprehension
squares=[i**2 for i in range(1,21) if i%2 == 0]
```

```python
# όλα τα ζευγάρια για ζυγό x∈[1,6], και μονό y∈[1,6]
pairs = [(x, y) for x in range(1, 7) for y in range(1, 7) if x%2 == 0 and y%2 == 1]
# pairs = [(2, 1), (2, 3), (2, 5), (4, 1), (4, 3), (4, 5), (6, 1), (6, 3), (6, 5)]
```

Συναρτήσεις

Μια συνάρτηση περιέχει ένα σύνολο από εντολές που εκτελούνται όταν κληθεί η συνάρτηση

- Οι παράμετροι έχουν τις τιμές που δίνονται κατά την κλήση της συνάρτησης
- μέσα στη συνάρτηση χρησιμοποιούμε το **return** για να επιστρέψουμε κάποια τιμή σε αυτόν που μας κάλεσε
- μπορούμε να επιστρέψουμε καμία, μία, ή περισσότερες τιμές

```python
# function definition
def name(parameter1, parameter2, ...):
    #code to be executed
    return expression

# function call
x = name(argument1, argument2, ...)
```

```python
def greet(name):
    print(f"Γειά σου, {name}!")

def add(x, y):
    return x + y

def add_and_multiply(x, y):
    sum = x + y
    product = x * y
    return sum, product

greet("Γιάννη")                          # δεν επιστρέφει τίποτα
sum = add(3, 5)                          # επιστρέφει 8
sum, product = add_and_multiply(3, 6)    # επιστρέφει 9, 18
```