Τελευταία ενημέρωση 4/10/2025

Python – Μέρος Α
ΕΚΔΔΑ

2.1 Τμηματικός Προγραμματισμός με την Python

Εβδομάδα 2/7

Σεπτέμβριος 2025

Τι είναι ο τμηματικός προγραμματισμός;

- Ο τμηματικός προγραμματισμός (modular programming) είναι ένας τρόπος προγραμματισμού που διευκολύνει την οργάνωση του κώδικα σε τμήματα, που (μπορούν να) είναι κατανοητά και εύκολα να συντηρηθούν
- Σε συνδυασμό με χρήση εύστοχων σχολίων και ευανάγνωστων ονομάτων αναγνωριστικών, ο τμηματικός προγραμματισμός μπορεί να οδηγήσει στην ανάπτυξη κώδικα υψηλής ποιότητας
- Στην Python τα τμήματα κώδικα μπορεί να είναι είτε ξεχωριστά αρχεία (modules), είτε συναρτήσεις (functions) μέσα σε αρχεία κώδικα
- Στην παρούσα παρουσίαση θα εξεταστούν οι συναρτήσεις ως τρόπος υλοποίησης του τμηματικού προγραμματισμού, ενώ στην επόμενη εβδομάδα θα εξεταστούν τα τμήματα κώδικα (module) και τα πακέτα (packages)

Συναρτήσεις

- Οι συναρτήσεις (functions) είναι τμήματα κώδικα στα οποία έχει αποδοθεί ένα όνομα, ορίζονται σε ένα σημείο του κώδικα και και μπορούν να καλούνται με το όνομα που τους έχει αποδοθεί από διάφορα σημεία του κώδικα, πολλές φορές
- Η κλήση και συνεπώς η εκτέλεση μιας συνάρτησης γίνεται χρησιμοποιώντας το όνομά της σε οποιοδήποτε σημείο κώδικα είναι ορατή
- Μια συνάρτηση δέχεται εισόδους (μπορεί και καμία), εκτελεί εντολές και επιστρέφει κάποια αποτελέσματα (μπορεί και κανένα) με τη χρήση της εντολής return

Η γενική μορφή του ορισμού μιας συνάρτησης είναι η ακόλουθη:

```
def όνομα_συνάρτησης(λίστα_παραμέτρων):
        εντολές
        return αποτέλεσμα
```

Σώμα συνάρτησης

Πλεονεκτήματα χρήσης συναρτήσεων

- Μια διάσημη τακτική επίλυσης σύνθετων προβλημάτων είναι η «Διαίρει και βασίλευε» (Divide & Conquer), όπου ένα πρόβλημα διασπάται σε άλλα μικρότερα υποπροβλήματα και αυτό συνεχίζεται μέχρι τα επιμέρους υποπροβλήματα να είναι απλά να επιλυθούν - με αυτό τον τρόπο καθίσταται εφικτή η επίλυση δύσκολων προβλημάτων
- Ένα παράπλευρο όφελος είναι ότι οι απαιτούμενες διορθώσεις και αλλαγές εντοπίζονται σε μικρότερα τμήματα κώδικα και αυτό διευκολύνει το έργο του προγραμματιστή
- Χρησιμοποιώντας συναρτήσεις η συγγραφή, κατανόηση, ανάγνωση και διόρθωση του κώδικα που επιλύει ένα πρόβλημα καθίσταται ευκολότερη

Παραδείγματα ορισμών και κλήσεων συναρτήσεων

Συνάρτηση που δεν επιστρέφει τιμή

```
# Συνάρτηση που δεν επιστρέφει τιμή
def print_greeting(name):
    print(f"Γεια σου {name}!")

# Κλήση συνάρτησης που δεν επιστρέφει τιμή
print_greeting("Μαρία")

result = print_greeting("Ελένη")
print("Αποτέλεσμα:", result)
```

```
Γεια σου Μαρία!
Γεια σου Ελένη!
Αποτέλεσμα: None
```

Συνάρτηση που επιστρέφει τιμή

```
# Συνάρτηση που επιστρέφει μια τιμή
def calculate_area(length, width):
    area = length * width
    return area

# Κλήση συνάρτησης calculate_area()
room_area = calculate_area(4.5, 3.2)
print(f"Το εμβαδόν είναι: {room_area} τ.μ.")
```

```
Το εμβαδόν είναι: 14.4 τ.μ.
```

Παράμετροι και ορίσματα συναρτήσεων

- Μια συνάρτηση στον ορισμό της μπορεί να δέχεται καμία, μια ή περισσότερες παραμέτρους (parameters) που ο σκοπός τους είναι να λειτουργήσουν ως είσοδοι τιμών που θα χρησιμοποιηθούν στο σώμα της συνάρτησης για την υλοποίηση της απαιτούμενης λειτουργικότητας
- Κατά την κλήση μιας συνάρτησης οι τιμές που δίνονται στις παραμέτρους ονομάζονται ορίσματα (arguments)
- Συνεπώς, οι παράμετροι μιας συνάρτησης είναι κατά τον ορισμό της συνάρτησης, ενώ τα ορίσματα κατά την κλήση της

```python
# Ορισμός συνάρτησης με 1 παράμετρο
def celsius_to_fahrenheit(celsius):
    fahrenheit = (celsius * 9/5) + 32
    return fahrenheit

# Κλήσεις συνάρτησης
temp1 = celsius_to_fahrenheit(0)      # Όρισμα: 0
temp2 = celsius_to_fahrenheit(100)    # Όρισμα: 100

print(f"0°C = {temp1}°F")
print(f"100°C = {temp2}°F")
```

```
0°C = 32.0°F
100°C = 212.0°F
```

Επιστροφή αποτελεσμάτων από συναρτήσεις

- Οι συναρτήσεις στην Python πάντα επιστρέφουν κάτι, αν μια συνάρτηση δεν επιστρέφει ρητά κάποια τιμή (με return) τότε επιστρέφει την τιμή None
- Είναι σύνηθες μια συνάρτηση να επιστρέφει περισσότερες από 1 τιμές, δείτε το παράδειγμα:

```
import math
```

χρειάζεται για το math.pi που επιστρέφει την τιμή του π που είναι κατά προσέγγιση 3.141592653589793

```
def circle_properties(radius):
    area = math.pi * radius**2
    circumference = 2 * math.pi * radius
    diameter = 2 * radius
    return area, circumference, diameter
```

unpacking

```
area, circum, diam = circle_properties(5)
print(f"Εμβαδόν: {area:.2f}, περίμετρος: {circum:.2f}, διάμετρος: {diam}")
```

```
Εμβαδόν: 78.54, περίμετρος: 31.42, διάμετρος: 10
```

Κλήση συναρτήσεων

- Στην Python υπάρχουν 2 τρόποι που μπορούμε να «περάσουμε» ορίσματα κατά την κλήση μιας συνάρτησης

  α) είτε να χρησιμοποιηθούν ορίσματα θέσης (positional arguments) που αντιστοιχίζονται σε παραμέτρους με βάση τη θέση τους

  β) είτε να χρησιμοποιηθούν ορίσματα λέξεων κλειδιών (keyword arguments) ή αλλιώς ονοματισμένα ορίσματα (named arguments), οπότε η σειρά εμφάνισης των ορισμάτων στην κλήση της συνάρτησης μπορεί να είναι οποιαδήποτε

- Μπορούν να χρησιμοποιηθούν και οι 2 τρόποι μαζί στην ίδια κλήση συνάρτησης

Παράδειγμα κλήσης συνάρτησης με ορίσματα θέσης και με ονοματισμένα ορίσματα

```
def calculate_rectangle_area(length, width):
    return length * width


r1 = calculate_rectangle_area(10, 20) # ορίσματα θέσης
r2 = calculate_rectangle_area(length=10, width=20) # ονοματισμένα ορίσματα
r3 = calculate_rectangle_area(width=20, length=10) # ονοματισμένα ορίσματα
print(f"{r1=}, {r2=}, {r3=}")
```

```
r1=200, r2=200, r3=200
```

Παράδειγμα κλήσης συνάρτησης με μίξη ορισμάτων θέσης και ονοματισμένων ορισμάτων

```
def process_order(item_name, quantity, price, tax_rate, express_shipping):
    total = quantity * price * (1 + tax_rate)
    if express_shipping:
        total += 10
    print(f"Παραγγελία: {quantity} x {item_name} = {total:.2f}€")


# Μόνο ορίσματα θέσης
process_order("Φορητός Η/Υ", 1, 999.99, 0.24, True)

# Μίξη ορισμάτων θέσης και ονοματισμένων ορισμάτων
process_order("Κινητό", 2, 599.99, express_shipping=True, tax_rate=0.24)
```

```
Παραγγελία: 1 x Φορητός Η/Υ = 1249.99€
Παραγγελία: 2 x Κινητό = 1497.98€
```

```
process_order(tax_rate=0.24, "Κινητό", 2, 599.99, express_shipping=True)
```

Η παραπάνω κλήση της process_order είναι λάθος διότι πρέπει να προηγούνται τα ορίσματα θέσης από τα ονοματισμένα ορίσματα κατά την κλήση, αλλιώς προκαλείται *SyntaxError: positional argument follows keyword argument*

Τεκμηρίωση συνάρτησης με συμβολοσειρές τεκμηρίωσης (1/2)

- Οι συμβολοσειρές τεκμηρίωσης (docstrings) είναι συμβολοσειρές πολλών γραμμών (συνεπώς περικλείονται σε τριπλά εισαγωγικά) που τοποθετούνται στην αρχή του σώματος των συναρτήσεων με σκοπό την περιγραφή της λειτουργίας των συναρτήσεων
- Η τοποθέτηση docstrings στις συναρτήσεις είναι προαιρετική, αλλά συνίσταται
- Από τη στιγμή που μια συνάρτηση διαθέτει docstring, μια κλήση της μορφής <όνομα_συνάρτησης>.__doc__ θα επιστρέψει το κείμενο του docstring της συνάρτησης <ονομα_συνάρτησης>
  - Το __doc__ είναι ένα παράδειγμα dunder, δηλαδή ενός αναγνωριστικού που ξεκινά και τελειώνει με 2 κάτω παύλες (double underscores)
- Αν κληθεί η help με όρισμα το όνομα της συνάρτησης θα εμφανιστεί το docstring σε περιβάλλον εμφάνισης βοήθειας (για έξοδο πρέπει να πατηθεί το πλήκτρο q)

Τεκμηρίωση συνάρτησης με συμβολοσειρές τεκμηρίωσης (2/2)

```
def calculate_area(radius):
    """
    Calculate the area of a circle given its radius.

    Parameters:
    radius (float): The radius of the circle.

    Returns:
    float: The area of the circle.
    """
    import math
    return math.pi * radius ** 2

print(calculate_area.__doc__)
```

```
Calculate the area of a circle given its radius.

  Parameters:
  radius (float): The radius of the circle.

  Returns:
  float: The area of the circle.
```

Η εντολή pass

- Κατά τη συγγραφή ενός προγράμματος μπορεί να έχει ανιχνευθεί η ανάγκη ύπαρξης μιας συνάρτησης αλλά να μην επιλεγεί η συγγραφή του σώματός της στη συγκεκριμένη φάση
- Τότε και προκειμένου να διατηρηθεί η συντακτική ορθότητα του κώδικα, γράφεται κανονικά η πρώτη γραμμή της συνάρτησης, αλλά ακολουθείται από την λέξη pass ή τρεις τελείες ... ακριβώς για να υποδηλωθεί ότι το σώμα της συνάρτησης θα συμπληρωθεί αργότερα
- Η εντολή pass είναι χρήσιμη για τη συγγραφή του «σκελετού» ενός κώδικα
- Επίσης, το pass μπορεί να χρησιμοποιηθεί στη θέση ενός μπλοκ κώδικα στις if, for, while που πρόκειται να υλοποιηθεί αργότερα

```
def foo():
    pass

def bar(a_parameter, another_parameter):
    ...

foo()
bar(1,2)
```

```
age = 20

if age >= 18:
    pass  # θα υλοποιηθεί αργότερα
else:
    print("Ανήλικος")
```

Προαιρετικές παράμετροι συναρτήσεων

- Οι παράμετροι των συναρτήσεων μπορούν είτε να είναι υποχρεωτικές να συμπληρωθούν για να είναι έγκυρη η κλήση της συνάρτησης, είτε να είναι προαιρετικές οπότε για κάθε προαιρετική παράμετρο ορίζεται μια προκαθορισμένη τιμή (default value) που χρησιμοποιείται αν δεν δοθεί αντίστοιχη τιμή ορίσματος

```
def calculate_interest(principal, rate=0.05, time=1, compound_frequency=1):
    amount = principal * (1 + rate / compound_frequency) ** (compound_frequency * time)
    interest = amount - principal
    return round(interest, 2)


# default τιμές: επιτόκιο 5%, 1 έτος, ετήσια εφαρμογή επιτοκίου
interest1 = calculate_interest(1000)
print(f"Τόκοι: {interest1}€")

# Προσαρμοσμένες τιμές: επιτόκιο 8%, 2 έτη, ετήσια εφαρμογή επιτοκίου
interest2 = calculate_interest(1000, rate=0.08, time=2)
print(f"Τόκοι: {interest2}€")
```

```
Τόκοι: 50.0€
Τόκοι: 166.4€
```

Άσκηση #1: Εκφώνηση

- Ορίστε τη συνάρτηση **calculate_total(price, quantity=1, discount=0)** που πολλαπλασιάζει την τιμή (price) με την ποσότητα (quantity), εφαρμόζει ένα ποσοστό έκπτωσης (discount) που είναι μια τιμή από 0 μέχρι 100 και επιστρέφει το συνολικό κόστος
- Καλέστε τη συνάρτηση για price = 75, quantity = 1, discount = 0 και εμφανίστε το αποτέλεσμα
- Καλέστε τη συνάρτηση για price = 75, quantity = 3, discount = 12 και εμφανίστε το αποτέλεσμα
- Καλέστε τη συνάρτηση για τιμές που δίνει ο χρήστης, πρώτα για την τιμή, μετά για την ποσότητα όπου αν ο χρήστης πατήσει enter χωρίς να εισάγει τιμή να χρησιμοποιείται η προκαθορισμένη τιμή και μετά για το ποσοστό έκπτωσης που αν ο χρήστης πατήσει enter χωρίς να εισάγει τιμή να χρησιμοποιείται επίσης η προκαθορισμένη τιμή

Άσκηση #1: Λύση

```python
def calculate_total(price, quantity=1, discount=0):
    """Υπολογισμός συνολικού κόστους λαμβάνοντας υπόψη τιμή, ποσότητα και έκπτωση."""
    subtotal = price * quantity
    total = subtotal * (1 - discount / 100)
    return total

print(f"{calculate_total(75):.2f}") # 75.00
print(f"{calculate_total(75, 3, 12):.2f}") # 198.00
price = float(input("Τιμή μονάδας: "))

quantity_input = input("Ποσότητα (enter για 1): ")
if quantity_input.strip() == "":
    quantity = 1
else:
    quantity = int(quantity_input)

discount_input = input("Έκπτωση (enter για 0): ")
if discount_input.strip() == "":
    discount = 0
else:
    discount = float(discount_input)

total_cost = calculate_total(price, quantity, discount)
print(f"Σύνολο: {total_cost}")
```

```
75.00
198.00
Τιμή μονάδας: 55
Ποσότητα (enter για 1):
Έκπτωση (enter για 0): 20
Σύνολο: 44.0
```

Συναρτήσεις με μεταβλητό πλήθος ορισμάτων

- Μια συνάρτηση μπορεί να να έχει μεταβλητό πλήθος ορισμάτων
- Για να συμβεί αυτό χρησιμοποιείται ο τελεστής * ή ο τελεστής ** που λειτουργούν ως τελεστές ανάπτυξης δομής
- Ειδικότερα ισχύουν τα:

  *args = μεταβλητό πλήθος ορισμάτων θέσης (positional arguments)

  **kwargs = μεταβλητό πλήθος ονοματισμένων ορισμάτων (keyword arguments)
- Τα ονόματα args και kwargs δεν είναι υποχρεωτικά, αλλά χρησιμοποιούνται αρκετά συχνά

Ορίσματα τύπου *args

- Η χρήση *args επιτρέπει την ευέλικτη χρήση ονοματισμένων ορισμάτων (positional arguments) που δεν έχουν προκαθοριστεί στον ορισμό της συνάρτησης
- Η ακόλουθη συνάρτηση μπορεί να κληθεί με οποιοδήποτε πλήθος ορισμάτων

```
def my_function(*args):
    print(f"Ελήφθησαν {len(args)} ορίσματα")
    for i, arg in enumerate(args):
        print(f"Όρισμα {i}: {arg}")


my_function()  # 0 ορίσματα
my_function(1)  # 1 όρισμα
my_function(1, 2, 3)  # 3 ορίσματα
my_function("a", "b", "c", "d")  # 4 ορίσματα
```

```
Ελήφθησαν 0 ορίσματα
Ελήφθησαν 1 ορίσματα
Όρισμα 0: 1
Ελήφθησαν 3 ορίσματα
Όρισμα 0: 1
Όρισμα 1: 2
Όρισμα 2: 3
Ελήφθησαν 4 ορίσματα
Όρισμα 0: a
Όρισμα 1: b
Όρισμα 2: c
Όρισμα 3: d
```

Ορίσματα τύπου **kwargs

- Η χρήση **kwargs επιτρέπει την ευέλικτη χρήση ονοματισμένων ορισμάτων (keyword arguments) που δεν έχουν προκαθοριστεί στον ορισμό της συνάρτησης
- Το kwargs είναι ένα λεξικό (θα μιλήσουμε για τα λεξικά στην εβδομάδα 4, για την ώρα αρκεί να γνωρίζουμε ότι ένα λεξικό αποτελείται από ζεύγη κλειδί: τιμή)

```
def my_function(**kwargs):
    print(f"Ελήφθησαν {len(kwargs)} ονοματισμένα ορίσματα")
    for key, value in kwargs.items():
        print(f"{key}: {value}")


my_function()  # 0 ορίσματα
my_function(name="Alice")  # 1 ονοματισμένο όρισμα
my_function(name="Bob", age=30)  # 2 ονοματισμένα ορίσματα
my_function(city="Ioannina", country="GR", \
    population=115000)  # 3 ορίσματα
```

```
Ελήφθησαν 0 ονοματισμένα ορίσματα
Ελήφθησαν 1 ονοματισμένα ορίσματα
name: Alice
Ελήφθησαν 2 ονοματισμένα ορίσματα
name: Bob
age: 30
Ελήφθησαν 3 ονοματισμένα ορίσματα
city: Ioannina
country: GR
population: 115000
```

Κανόνες για συνδυασμό args, kwargs, *args και **kwargs

- args = positional arguments (ορίσματα θέσης)
- kwargs = keyword arguments (ονοματισμένα ορίσματα)
- Κατά τον ορισμό συναρτήσεων:
  - Οι παράμετροι **kwargs, αν υπάρχουν, πρέπει πάντα να είναι τελευταίοι
- Κατά την κλήση συναρτήσεων:
  - Τα args πρέπει να βρίσκονται πριν τα kwargs
  - Σε συναρτήσεις με παραμέτρους μετά το *args, τα ορίσματα που τους δίνουν τιμές πρέπει να είναι kwargs

```
# def f1(**kwargs, x): # SyntaxError: invalid syntax
#     pass

def f2(x, y):
    print(f"{x=}, {y=}")

f2(1, 2) # x=1, y=2
f2(x=1, y=2) # x=1, y=2
f2(1, y=2) # x=1, y=2
# f2(1, x=2) # TypeError: f2() got multiple values for argument 'x'
# f2(x=1, 2) # SyntaxError: positional argument follows keyword argument


def f3(x, *args, y, **kwargs):
    print(f"{x=}, {args=}, {y=}, {kwargs=}")

f3(1, 2, 3, y=4, z=5, w=6) # x=1, args=(2, 3), y=4, kwargs={'z': 5, 'w': 6}
# f3(1, 2, 3, 4, z=5, w=6) # TypeError: f3() missing 1 required keyword-only argument: 'y'
```

Άσκηση #2: Εκφώνηση

- Ορίστε τη συνάρτηση evens_max(*args) που δέχεται οποιοδήποτε πλήθος ακέραιων ορισμάτων και επιστρέφει τη μέγιστη τιμή από τις άρτιες τιμές ορισμάτων ή -1 αν δεν υπάρχει όρισμα που να είναι άρτιο
- Καλέστε από το κύριο πρόγραμμα τη συνάρτηση evens_max για τα ακόλουθα ορίσματα και εμφανίστε το αποτέλεσμα:
  - 1, 2, 3, 4, 5
  - -10, -20, -30, -40
  - 1, 3, 5, 7, 9
- Να μη χρησιμοποιηθεί η built-in συνάρτηση max() της Python

Άσκηση #2: Λύση

```
def evens_max(*args):
    """Επιστρέφει τη μέγιστη τιμή από τα άρτια ορίσματα ή -1 αν δεν υπάρχει."""
    max_even = None
    for x in args:
        if x % 2 == 0:
            if max_even is None:
                max_even = x
            elif x > max_even:
                max_even = x

    if max_even is None:
        return -1
    else:
        return max_even


print(evens_max(1, 2, 3, 4, 5))         # 4
print(evens_max(-10, -20, -30, -40))    # -10
print(evens_max(1, 3, 5, 7, 9))         # -1
```

Υπάρχει συντομότερη pythonic λύση που χρησιμοποιεί τη built-in συνάρτηση max()

Εμβέλεια

- Η εμβέλεια (scope) μιας μεταβλητής έχει να κάνει με το εύρος του κώδικα στο οποίο η μεταβλητή είναι «γνωστή»
- Οι εμβέλειες της Python που θα εξετάσουμε είναι:
  - Τοπική (local)
  - Καθολική (global)
  - Μη τοπική (nonlocal)
- Για περισσότερα δείτε τι είναι ο κανόνας LEGB

Τοπικές μεταβλητές

- Οι μεταβλητές που δημιουργούνται μέσα σε μια συνάρτηση έχουν τοπική εμβέλεια (ονομάζονται τοπικές μεταβλητές), δηλαδή μπορούν να χρησιμοποιηθούν μόνο μέσα στη συνάρτηση
- Αν γίνει απόπειρα αναφοράς μιας μεταβλητής εκτός της συνάρτησης τότε θα προκληθεί NameError και η εκτέλεση του κώδικα θα διακοπεί με μήνυμα σφάλματος ότι το συγκεκριμένο όνομα είναι not defined

```
def greet():
    message = "Hello, world!"  # τοπική μεταβλητή που
    print(message) # χρησιμοποιείται μέσα στη συνάρτηση

greet()             # ✅
print(message)      # ❌ Error: NameError: name 'message'
                    # is not defined
```

```
Hello, world!
Traceback (most recent call last):
  File "/PYTHON-A/week2/functions_example11.py", line 6, in <module>
    print(message)   # ❌ Error: NameError: name 'message' is not defined
NameError: name 'message' is not defined
```

Καθολικές μεταβλητές

- Οι καθολικές μεταβλητές (global variables) δηλώνονται εκτός συναρτήσεων και είναι προσπελάσιμες οπουδήποτε στο πρόγραμμα (εντός και εκτός συναρτήσεων)
- Ωστόσο, για να τροποποιηθεί η τιμή μιας καθολικής μεταβλητής με εντολή που βρίσκεται μέσα σε μια συνάρτηση θα πρέπει η καθολική μεταβλητή να δηλωθεί μέσα στη συνάρτηση ως global
- Γενικά ισχύει ότι μέσα σε μια συνάρτηση οι εξωτερικές μεταβλητές μπορούν να αναγνωστούν αλλά δεν μπορούν να τροποποιηθούν (εκτός και αν δηλωθούν ως global ή nonlocal όπως θα δούμε στη συνέχεια)

```
# καθολική μεταβλητή
exchange_rate = 1.15  # EUR -> USD


def convert_to_usd(amount_eur):
    return amount_eur * exchange_rate


def update_rate(new_rate):
    global exchange_rate # πρέπει να υπάρχει
    exchange_rate = new_rate


print(convert_to_usd(50))
update_rate(1.2)
print(convert_to_usd(50))
```

```
57.49999999999999
60.0
```

Συναρτήσεις μέσα σε συναρτήσεις

- Η Python δίνει τη δυνατότητα να οριστεί μια συνάρτηση, εμφωλευμένα μέσα σε μια άλλη συνάρτηση, οπότε και ονομάζεται εμφωλιασμένη συνάρτηση (nested function) ή εσωτερική συνάρτηση (inner function)
- Η εσωτερική συνάρτηση μπορεί να αποτελεί βοηθητικό κώδικα που δεν θέλουμε να είναι ορατός εκτός της περικλείουσας (εξωτερικής) συνάρτησης, αλλά να χρησιμοποιείται από αυτή
- Οι εσωτερικές συναρτήσεις μπορούν να προσπελαύνουν μεταβλητές από την εξωτερική εμβέλεια
- Η εσωτερική συνάρτηση δημιουργείται κάθε φορά που καλείται η εξωτερική συνάρτηση
- Οι εσωτερικές συναρτήσεις συνδέονται με μια προχωρημένη προγραμματιστική έννοια που ονομάζεται κλειστότητα (closure), στην οποία όμως δεν θα αναφερθούμε

```
def total_price(base_price, tax_rate):
    # εσωτερική συνάρτηση: υπολογίζει το φόρο
    def tax(amount):
        return amount * tax_rate

    # κλήση εσωτερικής συνάρτησης
    tax_amount = tax(base_price)
    return base_price + tax_amount


print(total_price(100, 0.24))
print(total_price(50, 0.10))
```

```
124.0
55.0
```

Μη τοπικές μεταβλητές

- Οι μη-τοπικές (nonlocal) μεταβλητές μπορούν να εμφανιστούν σε περίπτωση συναρτήσεων με εσωτερικές συναρτήσεις
- Αν μια μεταβλητή σε μια εσωτερική συνάρτηση δηλωθεί ως nonlocal, αυτό σημαίνει ότι αναφέρεται σε μια μεταβλητή με αυτό το όνομα στην πλησιέστερη περικλείουσα συνάρτηση και μπορεί να τροποποιήσει την τιμή της
- Οι μη τοπικές μεταβλητές είναι χρήσιμες όταν υπάρχει μια μεταβλητή σε μια περικλείουσα συνάρτηση που θέλουμε να ενημερωθεί μέσα από την εσωτερική συνάρτηση

```
def shopping_cart():
    total = 0  # εξωτερική μεταβλητή για την add_items

    def add_items(prices):
        nonlocal total  # επιτρέπει τροποποίηση της total
        for price in prices:
            total += price
            print(f"+ {price}, νέο σύνολο {total}")

    add_items([10, 25, 5])
    print(f"Τελικό σύνολο: {total}")


shopping_cart()
```

```
+ 10, νέο σύνολο 10
+ 25, νέο σύνολο 35
+ 5, νέο σύνολο 40
Τελικό σύνολο: 40
```

Η δεσμευμένη λέξη assert

- Η δεσμευμένη λέξη assert δίνει τη δυνατότητα ελέγχου των ορισμάτων που περνάνε σε μια συνάρτηση έτσι ώστε αν ληφθούν μη επιτρεπτές τιμές να προκαλείται πρόωρος τερματισμός της εκτέλεσης του κώδικα
- Πρόκειται για μια μορφή προστασίας του κώδικα που επιβάλει την τήρηση περιορισμών έτσι ώστε μόνο εφόσον οι περιορισμοί ικανοποιούνται να προχωρά η εκτέλεση του κώδικα
- Με ένα προαιρετικό μήνυμα στην assert μπορεί να παρέχεται εξήγηση για το σφάλμα

```
def divide(a, b):
    """
    Διαίρεση δύο αριθμών, διασφαλίζοντας
    ότι ο διαιρέτης δεν είναι μηδέν.
    """
    assert b != 0, "Denominator must not be zero!"
    return a / b

print(divide(10, 2))
print(divide(5, 0))
```

```
5.0
Traceback (most recent call last):
  File "/PYTHON-A/week2/functions_example15.py", line 10, in <module>
    print(divide(5, 0))
  File "/PYTHON-A/week2/functions_example15.py", line 5, in divide
    assert b != 0, "Denominator must not be zero!"
AssertionError: Denominator must not be zero!
```

Άσκηση #3: Εκφώνηση

- Γράψτε ένα πρόγραμμα που να δημιουργεί τις καθολικές μεταβλητές temp_sum = 0, temp_count = 0, temp_max = None για άθροισμα, πλήθος και μέγιστη από θερμοκρασίες που έχουν καταγραφεί
- Ορίστε τη συνάρτηση add_temperature(temp) που με assert εξασφαλίζει ότι η θερμοκρασία temp είναι μεταξύ -30, 60 και ενημερώνει τις καθολικές μεταβλητές
- Ορίστε τη συνάρτηση average_temperature() που επιστρέφει το μέσο όρο των θερμοκρασιών και με assert εξασφαλίζει ότι έχει καταγραφεί τουλάχιστον μια θερμοκρασία
- Ορίστε τη συνάρτηση max_temperature() που επιστρέφει τη μέγιστη θερμοκρασία και με assert εξασφαλίζει ότι έχει καταγραφεί τουλάχιστον μια θερμοκρασία
- Καλέστε τις συναρτήσεις έτσι ώστε να προκληθεί μια φορά AssertionError και μια φορά να εμφανιστούν τα αποτελέσματα, μέσος όρος καταγεγραμμένων θερμοκρασιών, και μέγιστη καταγεγραμμένη θερμοκρασία

Άσκηση #3: Λύση

```
temp_sum, temp_count, temp_max = 0, 0, None

def add_temperature(temp):
    """Προσθέτει θερμοκρασία και ενημερώνει τις καθολικές μεταβλητές."""
    global temp_sum, temp_count, temp_max
    assert -30 <= temp <= 60, "Η θερμοκρασία πρέπει να είναι μεταξύ -30 και 60"
    temp_sum += temp
    temp_count += 1
    if temp_max is None or temp > temp_max:
        temp_max = temp

def average_temperature():
    """Επιστρέφει το μέσο όρο θερμοκρασιών."""
    assert temp_count > 0, "Δεν έχει καταγραφεί καμία θερμοκρασία"
    return temp_sum / temp_count

def max_temperature():
    """Επιστρέφει τη μέγιστη θερμοκρασία."""
    assert temp_count > 0, "Δεν έχει καταγραφεί καμία θερμοκρασία"
    return temp_max

# Προσθήκη θερμοκρασιών (έγκυρες)
add_temperature(20); add_temperature(25); add_temperature(18); add_temperature(30)
print("Μέσος όρος θερμοκρασιών:", average_temperature()) # 23.25
print("Μέγιστη θερμοκρασία:", max_temperature()) # 30
add_temperature(100)  # AssertionError
```

```
Μέσος όρος θερμοκρασιών: 23.25
Μέγιστη θερμοκρασία: 30
Traceback (most recent call last):
  File "/exercise.py", line 38, in <module>
    add_temperature(100)  # Αυτό θα προκαλέσει AssertionError
  File "/exercise.py", line 9, in add_temperature
    assert -30 <= temp <= 60, "Η θερμοκρασία πρέπει να είναι
μεταξύ -30 και 60"
AssertionError: Η θερμοκρασία πρέπει να είναι μεταξύ -30 και 60
```

Αναδρομή

- Μια ενδιαφέρουσα κατηγορία συναρτήσεων είναι οι αναδρομικές συναρτήσεις (recursive functions)
- Μια αναδρομική συνάρτηση καλεί τον εαυτό της και επαναλαμβάνει αυτή τη συμπεριφορά μέχρι να φτάσει στη λεγόμενη «βασική περίπτωση», οπότε και τερματίζεται η αναδρομή
- Γενικά μια αναδρομική συνάρτηση πρέπει να έχει:
  - Βασική περίπτωση: Μια συνθήκη που όταν θα συμβεί η συνάρτηση θα σταματήσει να καλεί τον εαυτό της
  - Αναδρομική περίπτωση: Το τμήμα που η συνάρτηση καλεί ξανά τον εαυτό της για ένα μικρότερο πρόβλημα
- Η αναδρομή (recursion) είναι χρήσιμη σε προβλήματα που μπορούν να διασπαστούν σε παρόμοια προβλήματα, όπως συμβαίνει στη διάσχιση δομών δεδομένων όπως τα δένδρα και τα γραφήματα

Παράδειγμα αναδρομικής συνάρτησης (1/2)

- Παραγοντικό: Το παραγοντικό ενός μη αρνητικού ακέραιου αριθμού, συμβολίζεται με το ! και ορίζεται ως εξής:
  - 0! = 1
  - n! = n * (n-1) * (n-2) * … * 1
- Μπορεί να οριστεί με πολύ φυσικό τρόπο με αναδρομική συνάρτηση που μοιάζει πολύ με τον ακόλουθο αναδρομικό μαθηματικό τύπο:
  - $n! = \begin{cases} 1 & n = 0 \\ n \times (n-1)! & n > 0 \end{cases}$

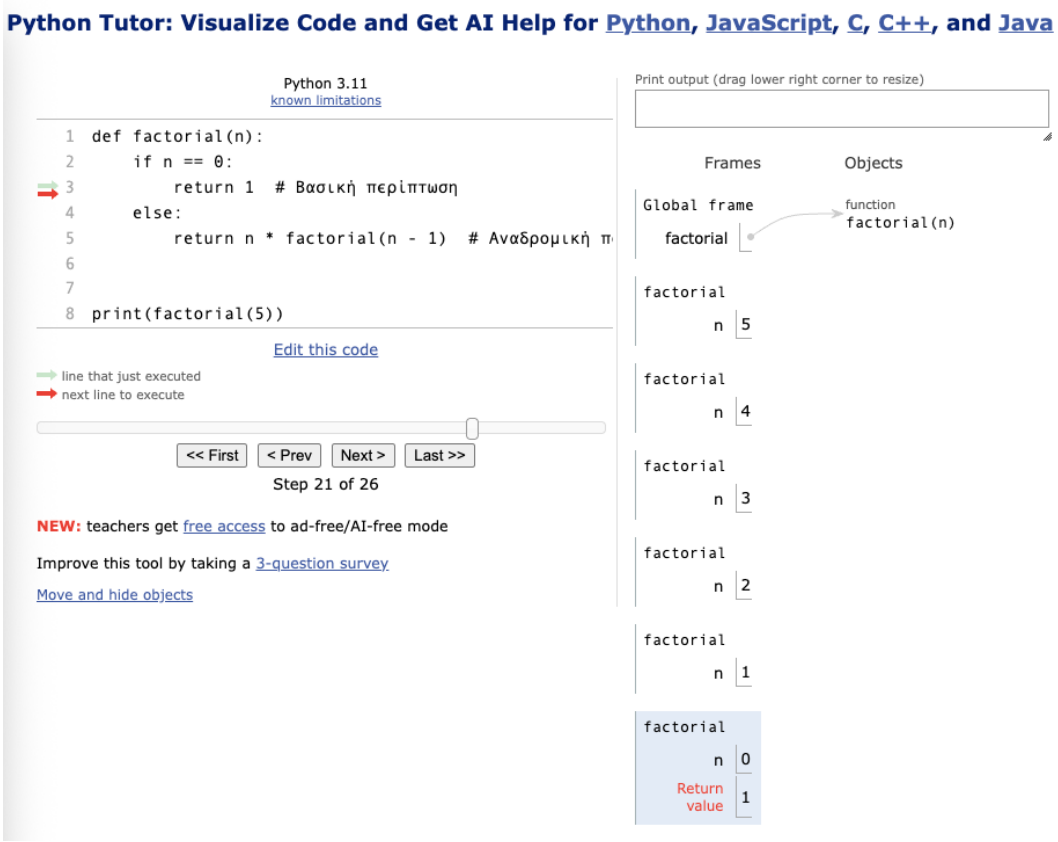
Παράδειγμα αναδρομικής συνάρτησης (2/2)

$$n! = \begin{cases} 1 & n = 0 \\ n \times (n-1)! & n > 0 \end{cases}$$

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

print(factorial(5))
print(factorial(10))
print(factorial(20))
```

```
120
3628800
2432902008176640000
```



https://pythontutor.com/

Άσκηση #4: Εκφώνηση

- Γράψτε μια αναδρομική συνάρτηση που να εξετάζει αν μια συμβολοσειρά είναι παλίνδρομη, διαβάζεται δηλαδή η ίδια από αριστερά προς τα δεξιά και από αριστερά προς τα δεξιά
- Καλέστε τη συνάρτηση για τις συμβολοσειρές RADAR, ΣΟΦΟΣ και PYTHON

Άσκηση #4: Λύση

```
def is_palindrome(s):
    if len(s) <= 1:
        return True
    if s[0] != s[-1]:
        return False
    return is_palindrome(s[1:-1])


s1 = "RADAR"
if is_palindrome(s1):
    print(f"{s1} είναι παλίνδρομη.")
else:
    print(f"{s1} δεν είναι παλίνδρομη.")

s2 = "ΣΟΦΟΣ"
if is_palindrome(s2):
    print(f"{s2} είναι παλίνδρομη.")
else:
    print(f"{s2} δεν είναι παλίνδρομη.")

s3 = "PYTHON"
if is_palindrome(s3):
    print(f"{s3} είναι παλίνδρομη.")
else:
    print(f"{s3} δεν είναι παλίνδρομη.")
```

```
RADAR είναι παλίνδρομη.
ΣΟΦΟΣ είναι παλίνδρομη.
PYTHON δεν είναι παλίνδρομη.
```

Τμηματικός προγραμματισμός vs. δομημένος προγραμματισμός

- Ο δομημένος προγραμματισμός (structured programming) είναι ένας τρόπος προγραμματισμού που στοχεύει στη βελτίωση της σαφήνειας και της ποιότητας του κώδικα χρησιμοποιώντας τις αλγοριθμικές δομές ακολουθία, επιλογή (π.χ. if/else) και επανάληψη (π.χ. for) αντί για της εντολές goto που χρησιμοποιούταν στον προγραμματισμό πολύ παλιότερα
- Ο δομημένος προγραμματισμός έχει να κάνει με το πως ο κώδικας εκτελείται ενώ ο τμηματικός προγραμματισμός με το πως ο κώδικας οργανώνεται σε διαφορετικά τμήματα
- Τα δομημένα προγράμματα είναι ευκολότερο να χωριστούν σε τμήματα

Τμηματικός προγραμματισμός vs. διαδικασιακός προγραμματισμός

- Ο διαδικασιακός προγραμματισμός (procedural programming) είναι ένας τρόπος προγραμματισμού όπου τα προγράμματα οργανώνονται σε ένα σύνολο διαδικασιών (συναρτήσεων) που λειτουργούν πάνω σε δομές δεδομένων
- Ο διαδικασιακός προγραμματισμός συχνά οδηγεί στον τμηματικό προγραμματισμό καθώς μια ή περισσότερες συσχετιζόμενες διαδικασίες (συναρτήσεις), μαζί με δεδομένα, οργανωμένα σε κατάλληλες δομές δεδομένων, μπορούν να οργανωθούν σε τμήματα (modules)

Τμηματικός προγραμματισμός vs. αντικειμενοστραφής προγραμματισμός

- Ο αντικειμενοστραφής προγραμματισμός (OOP = Object Oriented Programming) δίνει έμφαση μεταξύ άλλων στα αντικείμενα (object) και στην ενθυλάκωση (encapsulation) που είναι μια μορφή τμηματικού προγραμματισμού
- Στον OOP υπάρχει η έννοια της κλάσης που είναι ένα τμήμα κώδικα που εμπεριέχει τόσο δεδομένα όσο και συμπεριφορά
- Συνεπώς, υπάρχει σαφής συσχέτιση μεταξύ του τμηματικού προγραμματισμού και του OOP, καθώς ο OOP υλοποιεί τον τμηματικό προγραμματισμό με έναν συγκεκριμένο τρόπο
- Θα δούμε περισσότερα περί αντικειμενοστραφούς προγραμματισμού με την Python στην εβδομάδα 7

Προχωρημένες έννοιες σχετικές με συναρτήσεις

- Η Python διαθέτει προχωρημένα χαρακτηριστικά που σχετίζονται με τις συναρτήσεις
- Ορισμένα από αυτά είναι:
  - Λάμδα (lambda) συναρτήσεις ή ανώνυμες (anonymous) συναρτήσεις
  - Μερικές συναρτήσεις (partial functions)
  - Συναρτήσεις υψηλότερης τάξης (higher order functions)
  - Κλειστότητες (closures)
  - Γεννήτριες (generators)
  - Συρρουτίνες (coroutines)
  - Διακοσμητές (decorators)
- Στην παρούσα επιμόρφωση δεν θα ασχοληθούμε με τα παραπάνω, αλλά καλό είναι να γνωρίζουμε ότι υπάρχουν και καθιστούν την Python μια γλώσσα με πληθώρα προγραμματιστικών εννοιών στις οποίες μπορεί κανείς να εμβαθύνει για να βελτιώσει τον ποιότητα του κώδικά που γράφει

IDEs

- Δείτε τις διαφάνειες 11-13 από την παρουσίαση «1.2 Εισαγωγή στην εκπαιδευτική διαδικασία και στις απαιτήσεις του προγράμματος»
- Ένα IDE (Integrated Development Environment) είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης λογισμικού που επιτρέπει τη συγγραφή και εκτέλεση κώδικα, την αποσφαλμάτωση κώδικα (debugging), τη συγγραφή και εκτέλεση ελέγχων κώδικα (tests) κ.α.
- IDEs με μεγάλη αποδοχή στην κοινότητα των προγραμματιστών Python είναι το Visual Studio Code, το PyCharm και το Anaconda Spyder, αλλά είναι συχνό φαινόμενο μεμονωμένοι προγραμματιστές να έχουν κάποιο άλλο IDE ή άλλο περιβάλλον ανάπτυξης κώδικα (π.χ. vi, emacs) που προτιμούν

VS Code IDE

- Το VS Code είναι ένα ελαφρύ (lightweight) και ισχυρά επεκτάσιμο περιβάλλον για ανάπτυξη εφαρμογών σε Python, αλλά και σε άλλες γλώσσες προγραμματισμού
- Τα βασικά χαρακτηριστικά του VS Code που διευκολύνουν την ανάπτυξη κώδικα σε Python είναι:
  - Python extension: χρωματισμός κώδικα (syntax highlighting), αυτόματη συμπλήρωση κώδικα, άμεσος εντοπισμός σφαλμάτων (linting)
  - Debugger: βηματική εκτέλεση κώδικα, σημεία διακοπής εκτέλεσης (breakpoints), παρακολούθηση μεταβλητών (watches) κ.α.
  - Ενσωματωμένο τερματικό: εκτέλεση εντολών στο τερματικό χωρίς να φύγουμε από το VS Code
  - Διευκολύνσεις πλοήγησης στον κώδικα και refactoring (μαζικές αλλαγές στον κώδικα)
  - Υποστήριξη για ιδεατά περιβάλλοντα (virtual environments) που δημιουργούν ανεξάρτητες εγκαταστάσεις εκδόσεων της Python και βιβλιοθηκών
  - Υποστήριξη απευθείας εκτέλεσης Jupyter notebooks μέσα από το VS Code
  - Εγκατάσταση επιπλέον επεκτάσεων για την Python (π.χ. black, isort)
  - Version control για διατήρηση πλήρους ιστορικού αλλαγών στον κώδικα και συνεργασία σε ομάδες προγραμματιστών
  - Live Share: συνεργασία σε πραγματικό χρόνο με άλλους προγραμματιστές

VS Code Debugger

- Ο ακόλουθος κώδικας πρέπει να βρίσκει το άθροισμα των ακεραίων από το 1 μέχρι και το n, αλλά για n=5, επιστρέφει 10 αντί για 15
- Ποιο είναι το πρόβλημα; Ο debugger μπορεί να βοηθήσει!