ΠΕΡΙΕΧΟΜΕΝΑ:

Γιώργος Ν.

Ασημένιος Χορηγός Μαθήματος

Κωνσταντίνος Λ.

Ασημένιος Χορηγός Μαθήματος

- Οι **Εκφράσεις Λάμδα (Lambda Expressions)**:
  - είναι ανώνυμες **συναρτήσεις**
  - **τις οποίες μπορούμε να χειριστούμε σαν αντικείμενα**
    - π.χ. να διοχετεύσουμε ένα λάμδα σαν όρισμα σε μέθοδο, ή να αναθέσουμε ένα λάμδα σε μια μεταβλητή κ.λπ.

Συντακτικό ορισμού λάμδα:

```
lambda arguments: ret_value
```

- arguments:
  - ορίσματα, χωρισμένα με κόμματα.
- ret_value:
  - επιστρεφόμενη τιμή

Παράδειγμα 1: lambda_definition.py

```
addtwo = lambda x: x+2
print(addtwo(5))
print(addtwo(addtwo(1)))

# IIFE (Immediately Invoked Function Expression)
y = (lambda x: x*x)(3)
print(y)
```

Παρατηρήσεις

- Στο σώμα του lambda πρέπει να έχουμε **μία μόνο έκφραση**
  - (δεν επιτρέπεται π.χ. να έχουμε πολλές εντολές, δεν μπορούμε να ορίσουμε μεταβλητές, να χρησιμοποιήσουμε τις λέξεις class, def, return κ.ο.κ.)
- Τα ορίσματα ακολουθούν το συντακτικό των συναρτήσεων (θεσιακά, με λέξεις-κλειδιά, μεταβλητός αριθμός κ.ο.κ.)

- Η συνηθισμένη χρήση τους, είναι όταν θέλουμε να διοχετεύσουμε μια σύντομη συνάρτηση ως όρισμα και δεν θέλουμε να ορίσουμε μία ξεχωριστή συνάρτηση (γιατί βαριόμαστε ;-), έτσι είναι πιο σύντομο).

Παράδειγμα 2: lambda_arg_in_function.py

```
def f(x, y):
    return x(y)
print(f(lambda x: x*x, 6))
```

Σημείωση (βλ. βίντεο)

- Ο συναρτησιακός προγραμματισμός (functional programming) βασίζεται στο μοντέλο του λ-λογισμού (Church, 1930s) και είχε εκφραστεί αρχικά από γλώσσες όπως οι Lisp και η Haskell
- Οι δημοφιλείς γλώσσες (Python, Java κ.λπ.) ενσωμάτωσαν ιδέες που συνοψίζονται στα: (1) Μην αλλάζεις την κατάσταση των αντικειμένων στις συναρτήσεις (2) Δούλευε με immutable αντικείμενα.
- Τέτοιες ιδέες είδαμε π.χ. στις built-in: sorted(), min(), max() και τις map(), reduce(), filter() -> επόμενες διαφάνειες

- Ενδιαφέρον παρουσιάζει η χρήση του λαμδα σε εσωτερικές συναρτήσεις (που μπορεί να απλοποιήσει το συντακτικό που είδαμε για τα εργοστάσια συναρτήσεων - μάθ.14 και τη σύνταξη decorators - adv.μαθ.4 και 5)

Παράδειγμα 3: inner_lambda

```
def factory_power(power):
    def nth_power(number):
        return number ** power

    return nth_power


square = factory_power(2)
print(square(4))

cube = factory_power(3)
print(cube(4))
```

```
def factory_power(power):
    return lambda n: n ** power


square = factory_power(2)
print(square(4))

cube = factory_power(3)
print(cube(4))
```

Σημείωση:

- Το λάμδα «βλέπει» τις μεταβλητές που έχουν οριστεί στην εσωτερική συνάρτηση.
- Μία τέτοια συνάρτηση καλείται **closure** με τις γνωστές ιδιότητες των εσωτερικών συναρτήσεων.

Παράδειγμα 4: closure.py

```
def f(x, y):
    return lambda y: x ** y

g = f(2)

print(g(10))
print(g.__closure__)
print(g.__code__.co_freevars)
for item in g.__closure__:
    print(item.cell_contents)
```

- Μια λεπτή διαφορά στη λειτουργία φαίνεται στο παρακάτω παράδειγμα (βλ. βίντεο)

Παράδειγμα 5: closure2.py

```
l = []
for i in range(3):
    l.append(lambda: i)
for it in l:
    print(it())
```

- Η **built-in συνάρτηση map():**

```
map(function, iterable)
```

  - Εφαρμόζει τη συνάρτηση function, διαδοχικά, πάνω στα στοιχεία του iterable
  - Επιστρέφει iterator επί των διαδοχικών αποτελεσμάτων.

Παράδειγμα 6: map_example1

```
def square(x):
   return x*x
print(list(map(square, [1,2,3])))
# with lambda:
print(list(map(lambda x: x*x*x, [1,2,3])))
```

- Δηλαδή από τη map παίρνουμε ένα iterable με τα στοιχεία:
  - function(iterable[0])
  - function(iterable[1])
  - …

- Μπορούμε να διοχετεύσουμε και μεταβλητό αριθμό iterables:

```
map(function, *iterables)
```

- και επιστρέφονται (π.χ. για δύο iterables):
  - function(iterable1[0], iterable2[0])
  - function(iterable1[1], iterable2[1])
  - …

Παράδειγμα 7: map_example2.py

```
print(list(map(lambda x,y: x+y, [1,2,3], [4,5,6])))
print(dict(list(map(lambda x,y:
            (x,y), [1,2,3],
            ["one", "two", "three"]))))
print(list(map(lambda x,y,z:
        {"id": x, "name": y, "grade": z},
            (i for i in range(1000)),
            ["Bob", "Tom", "Pat"],
            [5,8,2])))
```

- Η map() μπορεί να προσομοιωθεί από comprehensions (και συνήθως θα προτιμάμε τα comprehensions, μιας και ο κώδικας είναι πιο pythonian)

Παράδειγμα 8: map_example3.py

```
print(list(map(lambda x: x*x*x, [1,2,3])))
print([x*x*x for x in [1,2,3]])

A=[1,2,3]
B=[4,5,6]
print(list(map(lambda x,y: x+y, [1,2,3], [4,5,6])))
print([A[i]+B[i] for i in range(len(A))])
```

- Η **built-in συνάρτηση filter():**

```
filter(function, iterable)
```

  - Εφαρμόζει τη συνάρτηση function, διαδοχικά, πάνω στα στοιχεία του iterable
    - Η συνάρτηση πρέπει να παίρνει ένα όρισμα και να επιστρέφει true/false
  - Επιστρέφει iterator με τα στοιχεία που ικανοποιούν την function (με όρισμα το στοιχείο, επιστρέφει true)

Παράδειγμα 9: filter_example1

```
def is_odd(x):
    return x%2 == 1

print(list(filter(is_odd, [1,2,3,4,5])))
print(filter(is_odd, [1,2,3,4,5]))

# with lambda:

print(list(filter(lambda x: x%2==0, [1,2,3,4,5])))
```

Παρατήρηση:

- Η filter() (όπως και η map()) μπορεί να προσομοιωθεί από comprehensions.

Παράδειγμα 10: filter_example2.py

```
def is_even(x):
    return x%2 == 0

my_list = [1,2,3,4,5]
print(list(filter(is_even, [1,2,3,4,5])))
print([x for x in my_list if x%2==0])
print([x for x in my_list if is_even(x)])
print([x for x in my_list if (lambda x: x%2==0)(x)])
```

Άσκηση 1:

Κατασκευάστε one-liner που να επιστρέφει από μία λίστα συμβολοσειρών, εκείνες που είναι παλινδρομικές.

- Με χρήση comprehensions
- Χωρίς χρήση comprehensions

- Η **συνάρτηση reduce() του functools:**

```
reduce(function, iterable, initializer=None)
```

  - Θα επιστρέψει μία ΤΙΜΗ που θα την υπολογίσει ως εξής:
    - Αρχικά ΤΙΜΗ = initializer
    - Επαναληπτικά ΤΙΜΗ = function(ΤΙΜΗ, iterable.next())
  - Αν initializer = None
    - Αρχικά ΤΙΜΗ = iterable[0]

Παρατήρηση:

- Η reduce μεταφέρθηκε στο functools στην Python 3.0
- Ωστόσο οι map-filter-reduce "πάνε πακέτο" και έτσι παρουσιάζεται εδώ.

Παράδειγμα 11: reduce_example1

```
from functools import reduce

# sum
print(reduce(lambda x, y: x+y, [1,2,3,4]))

# max
print(reduce(lambda x, y: x if x>y else y, [1,2,3,4]))

# sum of squares
print(reduce(lambda x, y: x + y*y, [1,2,3,4], 0))
```

Παρατήρηση:

- Η reduce() δεν μπορεί να προσομοιωθεί από comprehensions, όπως η map() και η filter()

Παράδειγμα 12: reduce_example2.py

```
from functools import reduce

students = [
   {
      "name": "Bob",
      "grade": 5,
   },
   ...
]

# students that passed
print(reduce(lambda x, y: x+[y["name"]] if y["grade"]>=5 else x,
students, []))

# get the entries with just the names of students that passed
print(reduce(lambda x, y: x+[{"name": y["name"]}] if
y["grade"]>=5 else x, students, []))

# student with lowest grade
print(reduce(lambda x, y: x if x["grade"] < y["grade"] else
y,students, (students[0])))
```