ΠΡΟΑΠΑΙΤΟΥΜΕΝΑ:

1. Python Advanced: Μάθημα 2 - Iterators
2. Python Advanced: Μάθημα 3 - Generators
3. Python Advanced: Μάθημα 6 - Λάμδα

ΠΕΡΙΕΧΟΜΕΝΑ:

1. Άπειροι και Συνδυαστικοί Iterators
2. Παραλλαγές της filter() και της map()
3. Συγχώνευση και Διάσπαση Iterators
4. Συσσώρευση και Ομαδοποίηση

Περίανδρος Μαυρομιχάλης

Σμαραγδένιος Χορηγός Μαθήματος

Μάνος Χ.

Ασημένιος Χορηγός Μαθήματος

ΜΑΘΗΜΑ 6.1: itertools

1. Άπειροι και Συνδυαστικοί Iterators

modules psounis

- Το **module itertools** περιέχει χρήσιμους, έτοιμους iterators.
- Πολλοί από αυτούς τους iterators είναι σχετικά απλοί, θα μπορούσαμε να τους υλοποιήσουμε και μόνοι μας, αλλά παρέχονται για τη διευκόλυνσή μας.

Iterators που παράγουν μία άπειρη ακολουθία τιμών:

iterator	Παράγει:
count(start[,step])	start, start+step, start+2step, …
cycle(iter)	iter[0],…,iter[n], iter[0],…,iter[n],…
repeat(obj[, times])	obj, obj, … (times φορες)

Παρατηρήσεις:

- Η count είναι άπειρη.
- Η cycle αποθηκεύει στη μνήμη τα στοιχεία του iter.
- Η repeat γίνεται άπειρη αν παραλείψουμε το όρισμα times.

Παράδειγμα 1: produce.values.py

```
from itertools import count, cycle, repeat
it = (val for val in count(10,2))
print(next(it), next(it))
it = (val for val in cycle([1,2,3,4]))
for i in range(10):
   print(next(it))
for i in repeat(10,3):
   print(i, end=" ")
```

Συνδυαστικοί Iterators:

- Iterators που προέρχονται από τη συνδυαστική:

iterator	Παράγει:
product(p,q,…)	Καρτεσιανό γινόμενο των iterators
product(A,repeat=n)	AxAx…xA (n φορές)
permutations(p[, r])	Διατάξεις p στοιχείων σε r θέσεις
combinations(p,r)	Συνδυασμοί p ανά r
combinations_with_replacement(p,r)	Συνδυασμοί p ανά r με επανάληψη

Παράδειγμα 2: combinatorial_iterators.py

```
from itertools import product, permutations, combinations

it_prod = product(["a","b","c"],[1,2,3])
print([x for x in it_prod])

it_perm = permutations([1,2,3],3)
for perm in it_perm:
   print(perm)

print("="*30)
it_comb = combinations([1,2,3,4],2)
for comb in it_comb:
   print(comb)
```

2. Παραλλαγές της filter() και της map()

Παραλλαγές της filter():

iterator	Παράγει:
dropwhile(predicate, iter)	iter[n], iter[n+1],... (αρχίζει όταν το pred. με όρισμα το στοιχείο του iter γίνει ψευδές)
takewhile(predicate, iter)	iter[0], iter[1],... (σταματα όταν το pred. με όρισμα το στοιχείο του iter γίνει ψευδές)
compress(iter, selectors)	Επιστρέφει τα στοιχεία του iter, που τα στοιχεία selectors στις αντίστοιχες θέσεις ερμηνεύονται σε true
filterfalse(predicate, iter)	Επιστρέφει τα στοιχεία του iter, που το lambda με όρισμα το στοιχείο είναι false.

Παραλλαγή της map():

iterator	Παράγει:
starmap (func, seq)	Διοχετεύει στη συνάρτηση func, τα ορίσματα που παράγονται από τον iterator seq

- Η func μπορεί να είναι πολλών ορισμάτων. Τότε θα πρέπει η seq να παράγει όλα τα ορίσματα σε μια ακολουθία, τα οποία θα χρησιμοποιηθούν ως ορίσματα.
- Επιστρέφει iterator με τα αποτελέσματα: func(seq[0]), func(seq[1]), ...

Παράδειγμα 3: filter.py

```
from itertools import dropwhile, takewhile, compress, filterfalse

elements = [1,2,3,4,5,1,2]
pred = lambda x: x<3
iterator = dropwhile(pred, elements)
for item in iterator:
    print(item)

print(list(takewhile(pred, elements)))

print(list(compress(elements, [1,1,1,1,0,0,1])))

print(list(filterfalse(lambda x:x%2==0, elements)))
```

Παράδειγμα 4: combinatorial_iterators.py

```
from itertools import starmap, product

def func(a,b,c,d):
    return abs(a-b)+abs(b-c)+abs(c-d)

iterator = starmap(func, product([1,2,3],[1,2,4],[3,5,2],[4,1,2]))
print(min(iterator))
```

Συγχώνευση Iterators:

iterator	Παράγει:
chain(p,q)	Επιστρέφει iterator που περιέχει πρώτα τα στοιχεία του p και έπειτα του q
zip_longest(p,q,…, fill_value=None)	Κατασκευάζει τον iterator (p[0],q[0]), (p[1],q[1]),… Αν κάποιος από αυτούς έχει μικρότερο μήκος από τους άλλους, τότε η θέση που του αντιστοιχεί συμπληρώνεται με την fill_value

Διάσπαση Iterators:

iterator	Παράγει:
tee(iter, n=2)	Επιστρέφει n ξεχωριστούς iterators, που ο καθένας περιέχει τα ίδια στοιχεια με τον iter.

Slicing iterator:

iterator	Παράγει:
islice(iter, [start,], stop, [,step])	Επιστρέφει το μέρος του iter από το start έως το stop-1 με βήμα step

Παράδειγμα 5: merging.py

```
from itertools import chain, zip_longest

a = (i for i in range(10))
b = (i for i in range(10,15))
c = (i for i in range(20,25))
for i in chain(a,b,c):
    print(i, end=" ")

print()
print(list(zip([1,2,3],[3,4],[5])))
print(list(zip_longest([1,2,3],[3,4],[5], fillvalue=0)))
```

Παράδειγμα 6: split.py

```
from itertools import tee, islice

iterator = (i for i in range(20))

it1, it2, it3, it4, it5, it6 = tee(iterator, 6)
print(list(it1))
print(list(it2))
print(list(it3))

print(list(islice(it4, 10)))
print(list(islice(it5, 5, 10)))
print(list(islice(it6, 5, 10, 2)))
```

Συσσώρευση Αποτελεσμάτων Διαπέρασης:

iterator	Παράγει:
accumulate(iter[,func])	Συσσωρεύει τα διαδοχικά αποτελέσματα άθροισης των στοιχείων του iter: iter[0], iter[0]+iter[1], iter[0]+iter[1]+iter[2],… Αν οριστεί το προαιρετικό όρισμα func, αυτό πρέπει να είναι μία συνάρτηση με δύο ορίσματα, ώστε να υπολογιστούν: res1 = iter[0] res2=func(res1, iter[1]) res3=func(res2, iter[2]), …

Ομαδοποίηση:

iterator	Παράγει:
groupby(iter)	Χωρίζει τα στοιχεία σε ομάδες με βάση την τιμή τους. Ίδιες διαδοχικές τιμές θα ανήκουν στην ίδια ομάδα. Κάθε ομάδα αποτελείται από το κλειδί (Την κοινή τιμή) και έναν iterator με τα στοιχεία της ομάδας
groupby (iter, func)	Ομοίως, αλλά η ομαδοποίηση γίνεται με βάση την τιμή της συνάρτησης με όρισμα την τιμή.

Παράδειγμα 7: accumulate_example.py

```
from itertools import accumulate

numbers = [1,2,3,4,5]
print(list(accumulate(numbers)))

it = accumulate(numbers, lambda x,y: x*y)
print(list(it))
```

Παράδειγμα 6: split.py

```
from random import randrange
from itertools import groupby

print("="*30)
numbers = [randrange(10) for i in range(20)]
print(numbers)
for k, items in groupby(numbers, lambda x: "even" if x%2==0
                                                else "odd"):
    print("Key: " + str(k), end=". Items: ")
    for item in items:
        print(item, end=" ")
    print()
```