Εισαγωγή στον Προγ/μό Υπολογιστών

Διάλεξη 10

Παράγωγες ακολουθίες

Περιεχόμενα

1. *Iterators*
   - *Iterables*
   - Σχέση με την εντολή `for`

2. *Generators* (γεννήτριες)
   - *Generator functions*
   - *Generator comprehensions*

Iterators

Iterators

- Οι επαναληπτικοί υπολογισμοί σαρώνουν ένα-ένα τα στοιχεία ενός δεδομένου που περιέχει πολλά στοιχεία, πχ, μιας *ακολουθίας* ή *συνόλου*

- ***Iterator***: δεδομένο το οποίο δίνει τη δυνατότητα σάρωσης όλων των στοιχείων με αφηρημένο τρόπο, δηλ., ανεξάρτητο από τον τύπο του δεδομένου που τα περιέχει

```
[1, 2, 'Hello', None]
```

Iterators

- Οι επαναληπτικοί υπολογισμοί σαρώνουν ένα-ένα τα στοιχεία ενός δεδομένου που περιέχει πολλά στοιχεία, πχ, μιας *ακολουθίας* ή *συνόλου*

- ***Iterator***: δεδομένο το οποίο δίνει τη δυνατότητα σάρωσης όλων των στοιχείων με αφηρημένο τρόπο, δηλ., ανεξάρτητο από τον τύπο του δεδομένου που τα περιέχει

1

```
[1, 2, 'Hello', None]
```

Iterators

- Οι επαναληπτικοί υπολογισμοί σαρώνουν ένα-ένα τα στοιχεία ενός δεδομένου που περιέχει πολλά στοιχεία, πχ, μιας *ακολουθίας* ή *συνόλου*

- ***Iterator***: δεδομένο το οποίο δίνει τη δυνατότητα σάρωσης όλων των στοιχείων με αφηρημένο τρόπο, δηλ., ανεξάρτητο από τον τύπο του δεδομένου που τα περιέχει

2

```
[1, 2, 'Hello', None]
```

Iterators

- Οι επαναληπτικοί υπολογισμοί σαρώνουν ένα-ένα τα στοιχεία ενός δεδομένου που περιέχει πολλά στοιχεία, πχ, μιας *ακολουθίας* ή *συνόλου*

- ***Iterator***: δεδομένο το οποίο δίνει τη δυνατότητα σάρωσης όλων των στοιχείων με αφηρημένο τρόπο, δηλ., ανεξάρτητο από τον τύπο του δεδομένου που τα περιέχει

```
'Hello'
```

```
[1, 2, 'Hello', None]
```

Iterators

- Οι επαναληπτικοί υπολογισμοί σαρώνουν ένα-ένα τα στοιχεία ενός δεδομένου που περιέχει πολλά στοιχεία, πχ, μιας *ακολουθίας* ή *συνόλου*

- ***Iterator***: δεδομένο το οποίο δίνει τη δυνατότητα σάρωσης όλων των στοιχείων με αφηρημένο τρόπο, δηλ., ανεξάρτητο από τον τύπο του δεδομένου που τα περιέχει

None

```
[1, 2, 'Hello', None]
```

Iterators

- Οι επαναληπτικοί υπολογισμοί σαρώνουν ένα-ένα τα στοιχεία ενός δεδομένου που περιέχει πολλά στοιχεία, πχ, μιας *ακολουθίας* ή *συνόλου*

- ***Iterator***: δεδομένο το οποίο δίνει τη δυνατότητα σάρωσης όλων των στοιχείων με αφηρημένο τρόπο, δηλ., ανεξάρτητο από τον τύπο του δεδομένου που τα περιέχει

```
{1, 2, 'Hello', None}
```

Iterators

- Οι επαναληπτικοί υπολογισμοί σαρώνουν ένα-ένα τα στοιχεία ενός δεδομένου που περιέχει πολλά στοιχεία, πχ, μιας *ακολουθίας* ή *συνόλου*

- ***Iterator***: δεδομένο το οποίο δίνει τη δυνατότητα σάρωσης όλων των στοιχείων με αφηρημένο τρόπο, δηλ., ανεξάρτητο από τον τύπο του δεδομένου που τα περιέχει

1

```
{1, 2, 'Hello', None}
```

Iterators

- Οι επαναληπτικοί υπολογισμοί σαρώνουν ένα-ένα τα στοιχεία ενός δεδομένου που περιέχει πολλά στοιχεία, πχ, μιας *ακολουθίας* ή *συνόλου*

- ***Iterator***: δεδομένο το οποίο δίνει τη δυνατότητα σάρωσης όλων των στοιχείων με αφηρημένο τρόπο, δηλ., ανεξάρτητο από τον τύπο του δεδομένου που τα περιέχει

2

```
{1, 2, 'Hello', None}
```

Iterators

- Οι επαναληπτικοί υπολογισμοί σαρώνουν ένα-ένα τα στοιχεία ενός δεδομένου που περιέχει πολλά στοιχεία, πχ, μιας *ακολουθίας* ή *συνόλου*
- ***Iterator***: δεδομένο το οποίο δίνει τη δυνατότητα σάρωσης όλων των στοιχείων με αφηρημένο τρόπο, δηλ., ανεξάρτητο από τον τύπο του δεδομένου που τα περιέχει

None

```
{1, 2, 'Hello', None}
```

Iterators

- Οι επαναληπτικοί υπολογισμοί σαρώνουν ένα-ένα τα στοιχεία ενός δεδομένου που περιέχει πολλά στοιχεία, πχ, μιας *ακολουθίας* ή *συνόλου*
- ***Iterator***: δεδομένο το οποίο δίνει τη δυνατότητα σάρωσης όλων των στοιχείων με αφηρημένο τρόπο, δηλ., ανεξάρτητο από τον τύπο του δεδομένου που τα περιέχει

```
'Hello'
```

```
{1, 2, 'Hello', None}
```

Iterators

- Οι επαναληπτικοί υπολογισμοί σαρώνουν ένα-ένα τα στοιχεία ενός δεδομένου που περιέχει πολλά στοιχεία, πχ, μιας *ακολουθίας* ή *συνόλου*

- ***Iterator***: δεδομένο το οποίο δίνει τη δυνατότητα σάρωσης όλων των στοιχείων με αφηρημένο τρόπο, δηλ., ανεξάρτητο από τον τύπο του δεδομένου που τα περιέχει

  - *Κατασκευαστής*: **`iter(`**`iterable`**`)`** 
    - Επιστρέφει *iterator* `iterator` για τη σάρωση στοιχείων του *iterable* `iterable`
      - *Iterable*, πχ., *ακολουθίες, σύνολα, λεξικά*

  - *Συνάρτηση επιλογής*: **`next(`**`iterator`**`)`** 
    - Επιστρέφει το επόμενο στοιχείο του `iterable` με το οποίο έχει κληθεί ο κατασκευαστής του `iterator`

Iterators

- Το τέλος της σάρωσης σηματοδοτεί το σφάλμα `StopIteration`

```
>>> t = iter([1, 2, 'Hello', None])
>>> next(t)
1
>>> next(t)
2
>>> next(t)
'Hello'
>>> print(next(t))
None
>>> next(t)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

Iterators

- Κάθε *iterator* διατηρεί τη δική του κατάσταση ανεξάρτητα από το όνομα που χρησιμοποιείται

```
>>> r = range(10, 20)
>>> t = iter(r)
>>> next(t)
10
>>> next(t)
11
>>> s = iter(r)
>>> next(s)
10
>>> next(t)
12
>>> u = t
>>> next(u)
13
```

Iterables

- Η `iter` είναι *γενική συνάρτηση*: η *κλήση* `iter(`**`iterable`**`)` επιστρέφει το αποτέλεσμα της *κλήσης*

  `iterable``.__iter__()`

- ***Iterable***: οποιοδήποτε *αντικείμενο* διαθέτει τη *μέθοδο* `__iter__`

```
>>> iterator = iter([1, 2, 'Hello', None])
```

ισοδύναμο με

```
>>> iterator = [1, 2, 'Hello', None].__iter__()
```

Iterators

- Ορισμένες ενσωματωμένες συναρτήσεις, πχ, `map`, `filter` επιστρέφουν *iterator*:

```
>>> result = map(lambda x: x*x, [1, -2, 3])
>>> next(result)
1
>>> next(result)
4
>>> next(result)
9
```

Iterators

- Η απεικόνιση των στοιχείων του *iterable* (2ο όρισμα της `map`) υπολογίζεται στην κλήση της `next` (*lazy evaluation*), όχι της `map`

```
>>> def square_n_print(x):
        print('--->', x, 'squared is', x*x)
>>> result = map(square_n_print, [1, -2, 3])
>>> next(result)
---> 1 squared is 1
>>> next(result)
---> 2 squared is 4
>>> next(result)
---> 3 squared is 9
```

Iterators

- Η ενσωματωμένη συνάρτηση `reversed` επιστρέφει *iterator* που διατρέχει στοιχεία ακολουθιών με ανάποδη σειρά

```
>>> t = reversed([1, 2, 3, 4])
>>> type(t)
<class 'list_reverseiterator'>
>>> next(t)
4
>>> next(t)
3
>>> next(t)
2
>>> next(t)
1
```

Iterators

- Η `next` είναι *γενική συνάρτηση*: η *κλήση* `next(`**`iterator`**`)` επιστρέφει την τιμή της *κλήσης*

`iterator.``__next__()`

- ***Iterator***: οποιοδήποτε *αντικείμενο* διαθέτει τη *μέθοδο* `__next__`

```
>>> t = iter([1, 2, 3, 4])
>>> next(t)
1
>>> t.__next__()
2
>>> t.__next__()
3
>>> next(t)
4
```

Σχέση με εντολή `for`

- Η εντολή `for` εσωτερικά κατασκευάζει και χρησιμοποιεί *iterator*

```
>>> for item in ls:
        print(item)
```

- Ισοδύναμο (εκτός ότι παράγεται σφάλμα `StopIteration` στο τέλος) με:

```
>>> iterator = ls.__iter__()
>>> while True:
        item = iterator.__next__()
        print(item)
```

Σχέση με εντολή `for`

- Η εντολή `for` λειτουργεί και για *iterators* – όχι μόνο για *iterables,* πχ.

```
>>> for item in reversed(ls):
        print(item)
```

- Ισοδύναμο (εκτός ότι παράγεται σφάλμα `StopIteration` στο τέλος) με:

```
>>> iterator = reversed(ls).__iter__()
>>> while True:
        item = iterator.__next__()
        print(item)
```

- Για να λειτουργήσουν οι *iterators* σε `for` loop, θα πρέπει να διαθέτουν μέθοδο `__iter__` η οποία επιστρέφει τον εαυτό τους

Iterators

- Οι *iterators* επιτρέπουν τη σάρωση *iterables* με τρόπο ανεξάρτητο από τον τύπο τους
  - Άθροιση στοιχείων λίστας με χρήση δεικτών (*χωρίς iterator*)

```
>>> def sum(ls):
        i, total = 0, 0
        while i < len(ls):
            total += ls[i]
            i += 1
        return total
>>> sum([1, 2, -5])
-1
```

  - Δεν λειτουργεί για δεδομένα που δεν είναι ακολουθίες:

```
>>> sum({1, 2, -5})
TypeError: 'set' object is not subscriptable
```

Iterators

- Οι *iterators* επιτρέπουν τη σάρωση *iterables* με τρόπο ανεξάρτητο από τον τύπο τους
  - Άθροιση στοιχείων με *iterator*

```
>>> def sum(a):
        total = 0
        for x in a:
            total += x
        return total
>>> sum([1, 2, 3, 4])
10
>>> sum({1, 2, 3, 4})
10
```

- Οι *iterators* βοηθούν στη συγγραφή *γενικών επαναληπτικών υπολογισμών*

Generators (γεννήτριες)

Generators

- Πολλές φορές θέλουμε να καθορίσουμε τον τρόπο που πραγματοποιείται η σάρωση στοιχείων, πχ. γιατί
  - Θέλουμε διαφορετική σειρά
  - Τα στοιχεία δεν βρίσκονται μέσα σε *iterable*
  - Τα στοιχεία δεν έχουν υπολογιστεί ακόμη (*lazy evaluation*)

➢ *Ορισμός iterator από τον προγραμματιστή*

Generators

- Πολλές φορές θέλουμε να καθορίσουμε τον τρόπο που πραγματοποιείται η σάρωση στοιχείων, πχ. γιατί
  - Θέλουμε διαφορετική σειρά
  - Τα στοιχεία δεν βρίσκονται μέσα σε *iterable*
  - Τα στοιχεία δεν έχουν υπολογιστεί ακόμη (*lazy evaluation*)

➢ ***Generators*** (γεννήτριες): *iterators* που όταν κληθεί η `next`, επιστρέφουν το επόμενο στοιχείο σύμφωνα με *υπολογισμό* που δίνεται από τον προγραμματιστή

1

```
x = x + 1
```

Generators

- Πολλές φορές θέλουμε να καθορίσουμε τον τρόπο που πραγματοποιείται η σάρωση στοιχείων, πχ. γιατί
  - Θέλουμε διαφορετική σειρά
  - Τα στοιχεία δεν βρίσκονται μέσα σε *iterable*
  - Τα στοιχεία δεν έχουν υπολογιστεί ακόμη (*lazy evaluation*)

➢ ***Generators*** (γεννήτριες): *iterators* που όταν κληθεί η `next`, επιστρέφουν το επόμενο στοιχείο σύμφωνα με *υπολογισμό* που δίνεται από τον προγραμματιστή

2

```
x = x + 1
```

Generators

- Πολλές φορές θέλουμε να καθορίσουμε τον τρόπο που πραγματοποιείται η σάρωση στοιχείων, πχ. γιατί
  - Θέλουμε διαφορετική σειρά
  - Τα στοιχεία δεν βρίσκονται μέσα σε *iterable*
  - Τα στοιχεία δεν έχουν υπολογιστεί ακόμη (*lazy evaluation*)

➢ ***Generators*** (γεννήτριες): *iterators* που όταν κληθεί η `next`, επιστρέφουν το επόμενο στοιχείο σύμφωνα με *υπολογισμό* που δίνεται από τον προγραμματιστή

3

```
x = x + 1
```

Generators

- Πολλές φορές θέλουμε να καθορίσουμε τον τρόπο που πραγματοποιείται η σάρωση στοιχείων, πχ. γιατί
  - Θέλουμε διαφορετική σειρά
  - Τα στοιχεία δεν βρίσκονται μέσα σε *iterable*
  - Τα στοιχεία δεν έχουν υπολογιστεί ακόμη (*lazy evaluation*)

- ***Generators*** (γεννήτριες): *iterators* που όταν κληθεί η `next`, επιστρέφουν το επόμενο στοιχείο σύμφωνα με *υπολογισμό* που δίνεται από τον προγραμματιστή

  - Κατασκευάζονται με τρεις τρόπους
    1. Με απευθείας υλοποίηση μεθόδου `__next__`
    2. Με συνάρτηση (*generator function*)
    3. Με έκφραση (*generator comprehension*)

Generators

- Γεννήτρια που διατρέχει τα πεζά λατινικά γράμματα `'a'` εως `'z'`

```
>>> class letters:
        def __init__(self):
            self.x = 'a'
        def __next__(self):
            val = self.x
            self.x = chr(ord(self.x) + 1)
            return val
        def __iter__(self):
            return self
>>> letter_generator = letters()
>>> next(letter_generator)
'a'
>>> next(letter_generator)
'b'
>>> next(letter_generator)
'c'
```

Generators

- ***Generator functions***: *Συναρτήσεις* που χρησιμοποιούν την εντολή `yield`. Δεν ερμηνεύονται από την Python ως κανονικές *συναρτήσεις*, αλλά χρησιμοποιούνται στην κατασκευή *generator*

```
>>> def letters():
        x = 'a'
        while x <= 'z':
            yield x
            x = chr(ord(x) + 1)
```

Χαρακτηριστικά:

1. Η κλήση τους *δεν* εκτελεί τις εντολές του σώματος

```
>>> letter_generator = letters()
>>> letter_generator
<generator object letters at 0x107217a20>
```

2. Οι εντολές εκτελούνται όταν κληθεί η `next` και αυτή επιστρέφει την τιμή της έκφρασης δίπλα στην εντολή `yield`

```
>>> next(letter_generator)
'a'
```

Generators

- ***Generator functions***: *Συναρτήσεις* που χρησιμοποιούν την εντολή `yield`. Δεν ερμηνεύονται από την Python ως κανονικές *συναρτήσεις*, αλλά χρησιμοποιούνται στην κατασκευή *generator*

```
>>> def letters():
        x = 'a'
        while x <= 'z':
            yield x
            x = chr(ord(x) + 1)
```

Χαρακτηριστικά:

3. Στην επόμενη κλήση της `next`, συνεχίζεται η εκτέλεση της εντολής μετά την προηγούμενη εντολή `yield` που εκτελέστηκε

```
>>> next(letter_generator)
'b'
```

4. Το *πλαίσιο* της πρώτης κλήσης της `next` διατηρείται για τις επόμενες (άρα διατηρούνται οι τιμές των ονομάτων)

Generators

- Γεννήτρια γραμμάτων

```
>>> def letters():
        x = 'a'
        while x <= 'z':
            yield x
            x = chr(ord(x) + 1)

>>> letter_generator = letters()
>>> next(letter_generator)
'a'
>>> next(letter_generator)
'b'
>>> next(letter_generator)
'c'
>>> next(letter_generator)
'd'
```

Generators

- ***Generator comprehension***: *έκφραση* της οποίας η τιμή είναι *generator* με συντακτικό παρόμοιο με αυτό των *list comprehensions*

- Παράδειγμα:
  - Κατασκευή λίστας γραμμάτων:

```
>>> letter_list = [chr(i) for i in range(ord('a'), ord('z') + 1)]
>>> letter_list
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm',
'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
```

  - Κατασκευή *generator*:

```
>>> letter_generator = (chr(i) for i in range(ord('a'),ord('z')+1))
>>> letter_generator
<generator object <genexpr> at 0x1073cb6d8>
>>> next(letter_generator)
'a'
```

Generators

- Παράδειγμα: άθροιση όρων ακολουθίας $1^2, 2^2, 3^2, \ldots, 100^2$:
  - Με λίστα:

```
>>> sum([x*x for x in range(1, 101)])
338350
```

    - Ολόκληρη η ακολουθία βρίσκεται στη μνήμη

  - Με *generator comprehension*

```
>>> sum((x*x for x in range(1, 101)))
338350
>>> sum(x*x for x in range(1, 101)) # το ίδιο
338350
```

    - Στη μνήμη βρίσκεται μόνο ένας όρος, ο οποίος αλλάζει σε κάθε κλήση της `next`