Προγραμματισμός Η/Υ

9ο Μάθημα

Γεννήτριες (generators), διαδρομές (path)

Διδάσκοντες

Νικόλαος Λαγαρός, Καθηγητής, Εργαστήριο Στατικής & Αντισεισμικών Ερευνών

☎ 210 772 2625, ✉ `nlagaros@central.ntua.gr`

Αθανάσιος Στάμος, ΕΔΙΠ, Τομέας Δομοστατικής

☎ 210 772 3665, ✉ `stamthan@central.ntua.gr`

Χριστόδουλος Φραγκουδάκης, ΕΔΙΠ, Κέντρο Ηλεκτρονικών Υπολογιστών

☎ 210 772 2434, ✉ `chfrag@central.ntua.gr`

Γεννήτριες (generators), διαδρομές (path)

Παραστάσεις Γεννήτριας (generator expressions)

- Η λίστα μπορεί να χρησιμοποιηθεί σε δομή επανάληψης αλλά μπορεί να δεσμεύει πολλή μνήμη χωρίς λόγο:

```
"Large lists may use large memory."
a = [i*3 for i in range(1000000)]
print(a)
print(sum(a))
import sys
print(sys.getsizeof(a))
```

- Την ίδια λειτουργικότητα έχουν και οι παραστάσεις γεννήτριας (generator expressions) χωρίς το κόστος μνήμης:

```
"Generators do not use large memory."
a = (i*3 for i in range(1000000))
print(a)
print(sum(a))
import sys
print(sys.getsizeof(a))
```

- Η γεννήτρια δεν υπολογίζει όλες τις τιμές στην αρχή, αλλά υπολογίζει την κάθε τιμή ακριβώς τη στιγμή που ζητείται (οκνηρός υπολογισμός).

Γεννήτριες (generators), διαδρομές (path)

Παραστάσεις Γεννήτριας (generator expressions)

- Αν υπάρχουν ήδη παρενθέσεις εκατέρωθεν της παράστασης γεννήτριας δεν χρειάζονται δεύτερες παρενθέσεις:

```
"Parentheses are not needed if enclosing parentheses exist: both commands do t
s = sum((i*3 for i in range(1000000)))
s = sum(i*3 for i in range(1000000))
print(s)
```

- Τα διάφορα αντικείμενα της python γενικώς δεν επιστρέφουν λίστες (ούτε γεννήτριες):

```
"Python objects in general return generators, not lists."
r = range(1, 100, 3)
print(r)
print(list(r))
```

Γεννήτριες (generators), διαδρομές (path)

Γεννήτριες (generators)

Οι γεννήτριες είναι συναρτήσεις που περιέχουν την εντολή yield:

```
"Generators are functions which contain the command yield."
def gen123():
    yield 1
    yield 2
    yield 3

g = gen123()
print(g)
for x in g:
    print(x)
```

Γεννήτριες (generators), διαδρομές (path)

Γεννήτριες (generators)

Μία γεννήτρια μπορεί να καλέσει μία άλλη μέσω του for:

```
"Nested generators are possible with the command for."
def gen123():
    yield 1
    yield 2
    yield 3

def gen1234():
    for q in gen123():
        yield q
    yield 4

for x in gen1234():
    print(x)
```

Γεννήτριες (generators), διαδρομές (path)

Γεννήτριες (generators)

Η κλήση γεννητριών είναι συνηθισμένη και μπορεί να γίνει και με την εντολή yield from:

```
"Nested generators are also possible with the command yield from."
def gen123():
    yield 1
    yield 2
    yield 3

def gen1234():
    yield from gen123()
    yield 4

for x in gen1234():
    print(x)
```

Γεννήτριες (generators), διαδρομές (path)

Γεννήτριες (generators)

Οι γεννήτριες είναι χρήσιμες για ακανόνιστες ακολουθίες:

```
"Generators are useful for irregular sequences."
def rangec(ia, ib, inc):
    "Like range but include ib."
    for i in range(ia, ib, inc):
        yield i
    yield ib

for x in rangec(0, 20, 3):
    print(x)
```

Γεννήτριες (generators), διαδρομές (path)

Γεννήτριες (generators)

Οι γεννήτριες είναι χρήσιμες και για πραγματικούς:

```
def frange(xa, xb, dx):
    "Like range but reals."
    if dx == 0.0: raise ValueError("Zero increment")
    x = xa
    if dx >= 0:
        while x < xb:
            yield x
            x += dx
    else:
        while x > xb:
            yield x
            x += dx

for a in frange(1.1, 2.2, 0.2):
    print(a)
print("-------------------------")
for a in frange(2.2, 1.1, -0.3):
    print(a)
print("-------------------------")
for a in frange(1.1, 2.2, 0.0):
    print(a)
```

Γεννήτριες (generators), διαδρομές (path)

Γεννήτριες

- Η συνάρτηση next() επιστρέφει το επόμενο στοιχείο μίας γεννήτριας:

```
def gen(): yield 12; yield 23
a = gen()
print(next(a))
print(next(a))
```

- Αν δεν υπάρχει επόμενο στοιχείο τότε καλείται η εξαίρεση StopIteration:

```
a = gen()
print(next(a))
print(next(a))
print(next(a))  #Καλείται StopIteration
```

- Για να μετατρέψουμε διάφορα αντικείμενα σε γεννήτριες (στην πραγματικότητα iterators) χρησιμοποιούμε τη συνάρτηση iter():

```
it = iter(range(2, 4))
print(next(it))
it = iter([10, 20])
print(next(it))
```

Γεννήτριες (generators), διαδρομές (path)

Πράξεις με γεννήτριες

Η βιβλιοθήκη itertools επιτρέπει τη σύνθεση γεννητριών (και άλλα):

```
import itertools as itt
for x in itt.repeat(3.14, 10):   #επαναλαμβάνει το 3.14 10 φορές
    print(x)

def g(): yield 10; yield 20
a = g()
b = range(1, 10)

for i in itt.cycle(b):        #επαναλαμβάνει την ακολουθία επ'άπειρο
    print(i)

for i in itt.chain(a, b):     #πρώτα την a, μετά τη b
    print(i)

for i in itt.accumulate(b):   #Αθροιστικά σύνολα
    print(i)
```

Γεννήτριες (generators), διαδρομές (path)

Πράξεις με γεννήτριες

```
for i in itt.islice(b, 5):    #μόνο τα πρώτα 5 στοιχεία της b
    print(i)

for i in itt.islice(itt.cycle(b), 15): #μόνο τα πρώτα 15 στοιχεία της b
    print(i)                           #επαναλαμβάνοντάς τη

a = [(1, 2), (3, 4, 5)]
for i in itt.chain.from_iterable(a): #Ισοπέδωση (flatten)
    print(y)

from math import sqrt
for i in map(sqrt, b):       #Υπολογισμός συνάρτησης
    print(i)

t = "NTUA"
for i in zip(t, b):          #Ζεύγη (πλειάδες) από τις 2 γεννήτριες (ή iterables)
    print(i)                 #Σταματάει όταν εξαντληθεί η 1η ή 2η γεννήτρια

t = "NTUA", "civil", "Python", "Programming", "4th"
for i in enumerate(t):     #Ζεύγη (πλειάδες) α/α και στοιχείων γεννήτριας
    print(i)
```

Γεννήτριες (generators), διαδρομές (path)

Διαδρομές αρχείων (path)

Η βιβλιοθήκη pathlib περιέχει το αντικείμενο Path που εκφράζει όνομα αρχείου ή φακέλου και την αντίστοιχη διαδρομή:

```
from pathlib import Path
a = Path("a.txt")
print(a)
```

Το αντικείμενο έχει διάφορες ιδιότητες:

```
a = a.resolve()   #Απόλυτη διαδρομή-το αρχείο πρέπει να υπάρχει
print(a)
d = Path(".").resolve()  #Τρέχων φάκελλος
print(d)
d = Path.cwd()    #Τρέχων φάκελλος
print(d)
p = a.parent      #Γονική διαδρομή
print(p)
n = a.name        #Όνομα αρχείου χωρίς διαδρομή
print(n)
s = a.suffix      #Κατάληξη αρχείου
print(s)
```

Γεννήτριες (generators), διαδρομές (path)

Διαδρομές αρχείων (path)

```
s = a.stem        #Ονομα αρχείου χωρίς κατάληξη
print(s)
b = p / "b.txt"   #Άλλο αρχείο στο γονικό φάκελλο του a
print(b)
b = a.with_name("b.txt") #Άλλο αρχείο στο γονικό φάκελλο του a
print(b)
c = b.with_suffix(".tex")   #Άλλο αρχείο στο γονικό φάκελλο του a
print(c)

fns = p.glob("*") #Ολα τα αρχεία και υποφάκελλοι του p
print(fns)        #γεννήτρια
print(list(fns))
print(" ".join(fn.name for fn in p.glob("*")))

fns = p.rglob("*") #Ολα τα αρχεία και υποφάκελλοι του p αναδρομικά
for fn in fns: print(fn)

print(a.exists())   #True αν το a υπάρχει
print(a.is_file())  #True αν το a είναι αρχείο
print(a.is_dir())   #True αν το a είναι φάκελλος
```

Γεννήτριες (generators), διαδρομές (path)

Διαδρομές αρχείων (path)

```
fns = (fn for fn in p.glob("*.txt") if fn.is_file())
for fn in fns: print(fn)  #Όλα τα αρχεία με κατάληξη .txt

dns = (fn for fn in p.rglob("*") if fn.is_dir())
for fn in dns: print(fn)  #Όλοι οι υποφάκελλοι αναδρομικά

a.rename(b)   #Μετονομασία αρχείου
print(" ".join(str(fn) for fn in p.glob("*.txt") if fn.is_file()))

b.unlink()    #Διαγραφή αρχείου
print(" ".join(str(fn) for fn in p.glob("*.txt") if fn.is_file()))

fw = a.open("w")    #Άνοιγμα αρχείου
fw.write("Ntua\n")
fw.close()
print(" ".join(str(fn) for fn in p.glob("*.txt") if fn.is_file()))
```

Γεννήτριες (generators), διαδρομές (path)

Διαδρομές αρχείων (path)

```
dn = p / "newdir"
dn.mkdir(parents=True)   #Δημιουργία φακέλλου
print(" ".join(str(fn) for fn in p.glob("*") if fn.is_dir()))

import os
os.chdir(str(dn))   #Αλλαγή τρέχοντος φακέλλου
x = Path("x.txt")
x.touch()           #Δημιουργία κενού αρχείου
print(x.resolve())
```

Ερωτήσεις;