Οικονομικό Πανεπιστήμιο Αθηνών, τμήμα Πληροφορικής
Μάθημα: Εισαγωγή στον Προγραμματισμό Υπολογιστών
Ακαδημαϊκό έτος: 2022–23
Διδάσκων: Α. Δημάκης

Τελική Εξέταση: διάρκεια 2 ώρες και 30 λεπτά

Σύνολο μονάδων: 11 (άριστα: 10)

ΚΑΛΗ ΕΠΙΤΥΧΙΑ!

Επίθετο:_____ **Όνομα:**_____ **ΑΜ:** _____

1η άσκηση. (1 μονάδα) Συμπληρώστε τα κενά με κώδικα Python ώστε να εμφανίζονται τα αναγραφόμενα αποτελέσματα.

Απάντηση:

```
>>> x = '*'
>>> x = x + '-' + x
>>> x = 3 * (x + '-') + x # Έκφραση χωρίς '*'
>>> print(x)
*-*-*-*-*-*-*
>>> z = '*'
>>> for x in range(2, 4):
        z = x * '*' + '-' + z +'-' + x * '*' # Έκφραση που περιέχει x
>>> print(z)
***-**-*-**-***
>>> f = lambda n: lambda s: n * s
>>> f(3)('*')
'***'
>>> f(2)('hello')
'hellohello'
```

2η άσκηση. (1) Τι θα εμφανίσει το ακόλουθο πρόγραμμα όταν εκτελεστεί;

```
def what_do_i_do(s):
    if len(s) == 1:
        print(s)
        return
    what_do_i_do(s[1:])
    print(s)
    what_do_i_do(s[-2::-1])

what_do_i_do('1234')


# Υπενθύμιση:
#>>> '123456789'[1:]
#'23456789'
#>>> '123456789'[-2::-1]
#'87654321'
```

Απάντηση:

```
4
34
3
234
2
32
3
1234
1
21
2
321
3
23
2
```

3η άσκηση. (2) Δώστε στο παρακάτω πλαίσιο πρόγραμμα που εμφανίζει στο τερματικό τα εξής:

```
*
**
***
****
1
22
333
0
22
444
6666
88888
```

Το πρόγραμμα σας θα πρέπει να αποφεύγει τη χρήση παρόμοιου κώδικα (πχ., πολλαπλές εντολές επανάληψης) και να είναι όσο το δυνατόν γενικότερο, δηλ., να εμφανίζονται τα παραπάνω δέντρα ή παραλλαγές τους με τη χρήση του ίδιου ακριβώς κώδικα χωρίς αλλαγές.

Απάντηση:

```
""" ΓΡΑΨΤΕ ΤΟ ΠΡΟΓΡΑΜΜΑ ΣΑΣ ΕΔΩ."""

def print_tree(n, func):
    for i in range(1, n + 1):
        print(i * func(i))

print_tree(4, lambda i: '*')
print_tree(3, lambda i: str(i))
print_tree(5, lambda i: str(2 * (i - 1)))


""" ΠΑΡΑΔΕΙΓΜΑ ΕΜΦΑΝΙΣΗΣ ΑΛΛΗΣ ΠΑΡΑΛΛΑΓΗΣ ΜΕ ΤΟΝ ΙΔΙΟ ΚΩΔΙΚΑ
>>> print_tree(10, lambda i: str(10 - i))
9
88
777
6666
55555
444444
3333333
22222222
111111111
0000000000

"""
```

4η άσκηση. Στην άσκηση αυτή θα δώσετε τρεις εναλλακτικές υλοποιήσεις της συνάρτησης `count_true` έτσι ώστε η `count_true(func, ls)` να επιστρέφει το πλήθος των στοιχείων `x` της λίστας `ls` για τα οποία η τιμή επιστροφής της κλήσης `f(x)` είναι `True`.

Για παράδειγμα,

```
>>> count_true(lambda x: x < 0, [-1, 3, -10, 5, -5, 7, 3])
3
>>> count_true(lambda x: x == 3, [1, 3, -10, 5, -5, 7, 3])
2
```

a) **(1)** *Υλοποίηση με επαναληπτικό υπολογισμό*: συμπληρώστε τον κώδικα που ακολουθεί ο οποίος πρέπει να περιέχει εντολές `for` ή `while` – χωρίς όμως *comprehensions** ή συναρτήσεις `map`, `filter`, `reduce`.

* *comprehensions είναι εκφράσεις της μορφής* `[x*x for x in range(1, 10)]`, `(x+1 for x in [1,2,3] if x % 1 == 1)`, *κτλ.*

Απάντηση:

```
def count_true(func, ls):
    """ΥΛΟΠΟΙΗΣΗ ΜΕ ΕΝΤΟΛΕΣ ΕΠΑΝΑΛΗΨΗΣ while ή for """
    """ΣΥΜΠΛΗΡΩΣΤΕ ΤΟ ΣΩΜΑ ΤΗΣ ΣΥΝΑΡΤΗΣΗΣ."""
    count = 0
    for x in ls:
        count += func(x)

    return count
```

b) **(1)** *Υλοποίηση με αναδρομή*: συμπληρώστε τον κώδικα που ακολουθεί ο οποίος πρέπει να περιέχει αναδρομικές κλήσεις χωρίς όμως εντολές επανάληψης `while`, `for` και εκφράσεις *comprehensions*.

Απάντηση:

```
def count_true(func, ls):
    """ΥΛΟΠΟΙΗΣΗ ΜΕ ΑΝΑΔΡΟΜΗ ΧΩΡΙΣ ΕΝΤΟΛΕΣ ΕΠΑΝΑΛΗΨΗΣ ΚΑΙ
    comprehensions"""
    """ΣΥΜΠΛΗΡΩΣΤΕ ΤΟ ΣΩΜΑ ΤΗΣ ΣΥΝΑΡΤΗΣΗΣ."""
```

```
    if len(ls) == 0:
        return 0
    else:
        return func(ls[0]) + count_true(func, ls[1:])
```

c) **(1)** *Υλοποίηση με επεξεργασία ακολουθίας*: Συμπληρώστε τον κώδικα που ακολουθεί ο οποίος πρέπει να περιέχει είτε *comprehensions* της Python (πχ, *list ή generator comprehensions*) ή τις ενσωματωμένες συναρτήσεις `map`, `reduce`, `filter` – χωρίς όμως εντολές επανάληψης `while`, `for` .

Απάντηση:

```
def count_true(func, ls):
    """ΥΛΟΠΟΙΗΣΗ ΜΕ COMPREHENSIONS Ή/ΚΑΙ ΕΝΤΟΛΕΣ map, reduce,
    filter"""
    """ΣΥΜΠΛΗΡΩΣΤΕ ΤΟ ΣΩΜΑ ΤΗΣ ΣΥΝΑΡΤΗΣΗΣ."""

    return sum(func(x) for x in ls)

"""ΕΝΑΛΛΑΚΤΙΚΕΣ ΑΠΑΝΤΗΣΕΙΣ (ΥΠΑΡΧΟΥΝ ΠΟΛΛΕΣ ΑΚΟΜΑ)
def count_true(func, ls):
    return sum(map(func, ls))


def count_true(func, ls):
    from functools import reduce
    return reduce(lambda x, y: x + y, map(func, ls))
"""
```

5η άσκηση.

a) **(1)** Συμπληρώστε στο παρακάτω πλαίσιο τον ορισμό της συνάρτησης `pack` ώστε η κλήση `pack(ls)` να μεταλλάσσει τα περιεχόμενα της λίστας `ls` ως εξής: εάν μια τιμή `x` εμφανίζεται `n` φορές στη λίστα `ls`, μετά την κλήση όλες οι εμφανίσεις της έχουν αντικατασταθεί με το *tuple* `(x, n)`. Η τιμή επιστροφής είναι `None`. (Τα tuples τοποθετούνται με οποιαδήποτε σειρά.)

Για παράδειγμα,

```
>>> ls = ['apple', 'hello', 'world', 'apple', 'hello', 'apple']
>>> pack(ls)
>>> ls
[('world', 1), ('apple', 3), ('hello', 2)]
```

Απάντηση:

```
"""ΔΩΣΤΕ ΤΟΝ ΟΡΙΣΜΟ ΤΗΣ ΣΥΝΑΡΤΗΣΗΣ pack."""

# ΛΥΣΗ ΜΕ ΕΠΕΞΕΡΓΑΣΙΑ ΑΚΟΛΟΥΘΙΑΣ
def pack(ls):
    ls[:] = list({x:ls.count(x) for x in set(ls)}.items())


"""
#ΛΥΣΗ ΜΕ ΕΠΑΝΑΛΗΠΤΙΚΟ ΥΠΟΛΟΓΙΣΜΟ ΜΕ ΣΥΝΟΛΟ
def pack(ls):
    new_ls = []
    for x in set(ls):
        new_ls.append((x, ls.count(x)))

    ls[:] = new_ls

#ΛΥΣΗ ΜΕ ΕΠΑΝΑΛΗΠΤΙΚΟ ΥΠΟΛΟΓΙΣΜΟ ΜΕ ΛΕΞΙΚΟ
def pack(ls):
    helper_dict = {}
    for x in ls:
        if x not in helper_dict:
            helper_dict[x] = ls.count(x)

    ls[:] = [(x, helper_dict[x]) for x in helper_dict]
"""
```

b) **(1)** Συμπληρώστε στο παρακάτω πλαίσιο τον ορισμό της συνάρτησης `unpack` η οποία έχει την αντίστροφη λειτουργία από την `pack` (και επίσης επιστρέφει `None`):

```
>>> ls
[('world', 1), ('apple', 3), ('hello', 2)]
>>> unpack(ls)
>>> ls
['world', 'apple', 'apple', 'apple', 'hello', 'hello']
```

(Η σειρά των στοιχείων της `ls` δεν έχει σημασία.)

Απάντηση:

```
"""ΔΩΣΤΕ ΤΟΝ ΟΡΙΣΜΟ ΤΗΣ ΣΥΝΑΡΤΗΣΗΣ unpack."""

#ΛΥΣΗ ΜΕ ΕΠΑΝΑΛΗΠΤΙΚΟ ΥΠΟΛΟΓΙΣΜΟ
def unpack(ls):
    new_ls = []
    for x, n in ls:
        new_ls += n * [x]

    ls[:] = new_ls


"""
#ΛΥΣΗ ΜΕ ΕΠΕΞΕΡΓΑΣΙΑ ΑΚΟΛΟΥΘΙΑΣ
def unpack(ls):
    from functools import reduce
    ls[:] = reduce(lambda a, b: a + [b[0]] * b[1], ls, [])

#ΛΥΣΗ ΜΕ ΑΝΑΔΡΟΜΗ
def unpack(ls, i = 0):
     if i == len(ls):
         return
     else:
        x, n = ls[i]
        ls[i:i + 1] = n * [x]
        unpack(ls, i + n)
"""
```

6η άσκηση. (2) Εδώ θα υλοποιήσετε μεταλλασσόμενο (*mutable) δεδομένο* που αναπαριστά σημείωμα.

Ο χειρισμός γίνεται μέσω των συναρτήσεων:

- `make_note(title)`: (*συνάρτηση κατασκευαστή* που*)* επιστρέφει δεδομένο που αναπαριστά σημείωμα με τίτλο που δίδεται από τη συμβολοσειρά `title`.
- `append(note, text)`: προσθέτει το κείμενο `text` στο τέλος του σημειώματος `note` και επιστρέφει συμβολοσειρά που περιέχει τον τίτλο και το σύνολο του κειμένου που έχει προστεθεί.

Για παράδειγμα,

```
>>> psonia = make_note('Ψώνια: ')
>>> append(psonia, '1. ψωμί')
'Ψώνια: 1. ψωμί'
>>> append(psonia, '2. χυμό')
'Ψώνια: 1. ψωμί2. χυμό'
>>>
>>> note2 = make_note('Μην ξεχάσω: ')
>>> append(note2, 'τηλεφωνήσω Μαρία')
'Μην ξεχάσω: τηλεφωνήσω Μαρία'
>>> append(psonia, '3. μπαταρίες')
'Ψώνια: 1. ψωμί2. χυμό3. μπαταρίες'
```

Υλοποιήστε τις παραπάνω συναρτήσεις στο πλαίσιο που ακολουθεί. (Μπορείτε να δώσετε επίσης όσους επιπλέον ορισμούς θεωρήσετε αναγκαίο.)

Απάντηση:

```
""" ΣΥΜΠΛΗΡΩΣΤΕ ΚΩΔΙΚΑ ΕΔΩ."""

def make_note(title):
    return [title]

def append(note, text):
    note[0] += text
    return note[0]


"""ΕΝΑΛΛΑΚΤΙΚΗ ΑΠΑΝΤΗΣΗ

def make_note(title):
    class Note:
        def __init__(self):
            self.text = title
        def append(self, text):
            self.text += text
            return self.text
    return Note()

def append(note, text):
    return note.append(text)

"""
```