סיכום למבחן – שפות תכנות

קרן רצ'ב

Generator

```
def myGenerator1(n):
    for i in range(n):
        yield i
def myGenerator2(n, m):
    for j in range(n, m):
        yield j
def myGenerator3(n, m):
    yield from myGenerator1(n)
    yield from myGenerator2(n, m)
    yield from myGenerator2(m, m+5)
a = myGenerator1(5)
print(a.__next__())
print(a.__next__())
print(a.__next__())
print(list(myGenerator2(5, 10)))
print(list(myGenerator3(0, 10)))
print(list(myGenerator3(0, 10)))
```

מימוש בעזרת פונקציה

```
0
1
2
[5, 6, 7, 8, 9]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

```
g = (x for x in range(10))
print(list(g))
```

```
$ python generator_example_4.py
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Sending Values to Generators

- The syntax is send() or send(value). Without any value, the send method is equivalent to a next() call. This method can also use None as a value. In both cases, the result will be that the generator advances its execution to the first yield expression.

GENERATOR במימוש כמחלקה, ממומש בדיוק כמו איטרטור.

DECORATOR

```
def dec(func):
    def wrapper(*args, **kwargs):
        targs = list(args)
        for i,st in enumerate(targs):
            if isinstance(st, str):
                targs[i] = str.upper(st)
        ttargs=tuple(targs)
        tkwargs=kwargs
        for i in tkwargs.keys():
            if isinstance(tkwargs[i], str):
                tkwargs[i] = tkwargs[i].upper
            return(func(ttargs, tkwargs))
    return wrapper
```

DICTIONARY

TUPLE- INMUTABLE

ניתן לראות, שבשביל לשנות את הערך של args, יש צורך במשתנה עזר, כי tuple הוא inmutable. תוכנית זו בודקת כול איבר, אם הוא String היא הופכת אותו לאותיות גדולות.

על מנת לספור שינוים שנעשו\ כמות פעמים שהדקרטור נקרא סה"כ\ עבור כול פונקציה :

סה"כ- יש לשמור משתנה גלובלי , ב- main, ובתוך הwrapper יש להגדירו כ- global.

עבור כול פונקציה עטופה- יש לשמור משתנה מתחת לdec, ומחוץ לפונק' wrapper, ובתוך הwrapper יש להגדירו כ-nonlocal

ITERATOR

```
class Counter(object):
    def __init__(self, low, high):
        self.current = low
        self.high = high
    def __iter__(self):
        'Returns itself as an iterator object'
        return self
    def __next__(self):
        'Returns the next value till current is lower than high'
        if self.current > self.high:
            raise StopIteration
        else:
            self.current += 1
        return self.current - 1

c = Counter(5,10)
for i in c:
    print(i, end=' ')
c = Counter(5,10)
print()
print(next(c))
print(next(c))
print(next(c))
print(next(c))
```

examEx

```
5 6 7 8 9 10
5
6
7
8
```

דוגמא ל Decorator שמחזיר כמה פונקציות לא החזירו ערכים, כמה החזירו ערך אחד וכמה החזירו יותר מערך אחד. וגם כמה פעמים קראו לכול פונקציה.

```
def dec2(func, num=0):
    dicalls = {0:0, 1:0, 2:0}
    def wrapper(*args, **kwargs):
        nonlocal dicalls
        nonlocal  num
        num += 1
        print("func ", func.__name__, "was called ", num, "times")
        retVal = func(*args, **kwargs)
        if retVal == None:
            dicalls[0]+=1
        elif isinstance(retVal, tuple):
            dicalls[2] += 1
        else:
            dicalls[1] += 1
        print(dicalls)
        return retVal
    return wrapper
```

בדיקה של סודוקו:

```
def Is_sudoko(x):
    for i in x:
        if len(set(i))!=9:
            return False
    for i in range(9):
        if len(set([x[a][i] for a in range(9)]))!=9:
            return False
    for i in range(0,9,3):
        for j in range(0,9,3):
            temp=x[i][j:j+3]+x[i+1][j:j+3]+x[i+2][j:j+3
            if len(set(temp))!=9:
                return False
    return True
```

LIST

```
mylist = ["apple", "banana", "cherry"]
```

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Converting list of tuples to dictionary:

```
# initializing the list
tuples = [('Key 1', 1), ('Key 2', 2), ('Key 3', 3), ('Key 4', 4), ('Key 5', 5)]

# converting to dict
result = dict(tuples)

# printing the result
print(result)
```

TUPLE

```
mytuple = ("apple", "banana", "cherry")
```

Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

Join two tuples:

```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)
```

Convert the tuple into a list to be able to change it:

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)

print(x)
```

Tuples allow duplicate values

Ordered

When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.

Unchangeable

Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

SET

```
myset = {"apple", "banana", "cherry"}
```

Set items are unordered, unchangeable, and do not allow duplicate values.

Once a set is created, you cannot change its items, but you can add new items.

Check if "banana" is present in the set:

```
thisset = {"apple", "banana", "cherry"}

print("banana" in thisset)
```

Add an item to a set, using the `add()` method:

```
thisset = {"apple", "banana", "cherry"}

thisset.add("orange")

print(thisset)
```

Add elements from `tropical` into `thisset` :

```
thisset = {"apple", "banana", "cherry"}
tropical = {"pineapple", "mango", "papaya"}

thisset.update(tropical)

print(thisset)
```

Remove "banana" by using the `remove()` method:

```
thisset = {"apple", "banana", "cherry"}

thisset.remove("banana")

print(thisset)
```

The object in the `update()` method does not have to be a set, it can be any iterable object (tuples, lists, dictionaries etc.).

DICT

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
```

Get the value of the "model" key:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict["model"]
```

Get a list of the keys:

```
x = thisdict.keys()
```

Loop through both *keys* and *values*, by using the `items()` method:

```
for x, y in thisdict.items():
  print(x, y)
```

You can also use the `values()` method to return values of a dictionary:

```
for x in thisdict.values():
  print(x)
```

You can use the `keys()` method to return the keys of a dictionary:

```
for x in thisdict.keys():
  print(x)
```

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Reduce

- At first step, first two elements of sequence are picked and the result is obtained.
- Next step is to apply the same function to the previously attained result and the number just succeeding the second element and the result is again stored.
- This process continues till no more elements are left in the container.
- The final returned result is returned and printed on console.

```python
# python code to demonstrate working of reduce()

# importing functools for reduce()
import functools

# initializing list
lis = [1, 3, 5, 6, 2, ]

# using reduce to compute sum of list
print("The sum of the list elements is : ", end="")
print(functools.reduce(lambda a, b: a+b, lis))

# using reduce to compute maximum element from list
print("The maximum element of the list is : ", end="")
print(functools.reduce(lambda a, b: a if a > b else b, lis))
```

Output

```
The sum of the list elements is : 17
The maximum element of the list is : 6
```

Map

The `map()` function executes a specified function for each item in an iterable. The item is sent to the function as a parameter.

Returns a Map\List, depends on the python version

```
<map object at 0x034244F0>
['appleorange', 'bananalemon', 'cherrypineapple']
```

```python
def myfunc(a, b):
  return a + b

x = map(myfunc, ('apple', 'banana', 'cherry'), ('orange', 'lemon', 'pineapple'))
print(x)

#convert the map into a list, for readability:
print(list(x))
```

```python
def myfunc(n):
  return len(n)

x = map(myfunc, ('apple', 'banana', 'cherry'))
```

Filter

The `filter()` function returns an iterator were the items are filtered through a function to test if the item is accepted or not.

Filter the array, and return a new array with only the values equal to or above 18:

```
ages = [5, 12, 17, 18, 24, 32]

def myFunc(x):
  if x < 18:
    return False
  else:
    return True

adults = filter(myFunc, ages)

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

Enumerate

A lot of times when dealing with iterators, we also get a need to keep a count of iterations. Python eases the programmers' task by providing a built-in function enumerate() for this task.

Enumerate() method adds a counter to an iterable and returns it in a form of enumerate object. This enumerate object can then be used directly in for loops or be converted into a list of tuples using list() method.

```
# Python program to illustrate
# enumerate function in loops
l1 = ["eat","sleep","repeat"]

# printing the tuples in object directly
for ele in enumerate(l1):
    print (ele)
print
# changing index and printing separately
for count,ele in enumerate(l1,100):
    print (count,ele)
```

```
(0, 'eat')
(1, 'sleep')
(2, 'repeat')

100 eat
101 sleep
102 repeat
```

```
# Without changing index and printing separately
for count,ele in enumerate(l1):
    print (count,ele)
```

```
0 eat
1 sleep
2 repeat
```

__CALL__

Python has a set of built-in methods and `__call__` is one of them. The `__call__` method enables Python programmers to write classes where the instances behave like functions and can be called like a function. When the instance is called as a function; if this method is defined, `x(arg1, arg2, ...)` is a shorthand for `x.__call__(arg1, arg2, ...)`.

```
class a():
    y=0
    def __init__(self,y):
        self.y =y
    def __call__(self,z):
        if z > self.y:
            return z-self.y
        else:
        return self.y-z
class b(a):
    def __call__(self, z=4):
        if z > self.y:
            return z-self.y
        else:
            return self.y-z
print(a(5)(b(6)()))
print(a(6)(b(5)(6)))
```

Output -
```
3
5
```

```
class Example:
    def __init__(self):
        print("Instance Created")

    # Defining __call__ method
    def __call__(self):
        print("Instance is called via special method")

# Instance created
e = Example()

# __call__ method will be called
e()
```

Output :

```
Instance Created
Instance is called via special method
```

List comprehension examples

ב. (10 נק') כתוב באמצעות list comprehension ביטוי המקבל רשימה של tuples. כל **tuple (i,x)**: מורכב משני מספרים שלמים חיוביים המספר הראשון (i) הוא האינדכס והמספר השני (x) הוא הערך. יש להרכיב רשימה חדשה בה כל ערך נמצא באינדקס שלו. יש לבדוק שקיימים כל האינדקסים מ-0 עד האינדקס הגדול ביותר. במידה וחסר אינדקס, יש להשלים את הערך במיקום החסר עם המספר 1000-.

לדוגמא, עבור הקלט [(4,9),(0,2),(1,4),(3,2)] יחזיר הביטוי את הרשימה [2 .4 . -1000 .2. 9] (האינדכס 2 חסר ולפיכך 1000 -)

```
lst = [(4,9),(0,2),(1,4),(3,2)]
x = [dict(lst)[h1] if h1 in dict(lst).keys() else h2 for h1,h2 in[enumerate([-1000 for u in range(reduce(max, [x for x,y in lst])+1)])]]
print(x)
```

כדאי לשים לב לשימוש היפה כאן בהפיכת הרשימה של ה tuples ל- dictionary מה שנותן עבר קל יותר על ה-keys.

<u>קבלת רשימה של מספרים והחזרת רשימה חדשה של strings שבה רשום עבור כול מספר אם הוא ראשוני או לא</u>

```
n=10
m=20
numbers = range(n,m+1)
M = ["Prime"  if all(x % y != 0 for y in range(2, x)) else "NotPrime" for x in numbers]
print(M)
```

דוגמא לGenerator הממומש כמחלקה וכפונקציה, שמחזיר את כול המספרים הראשוניים עד מספר מסוים

```
def primeNumbersGenerator(MaxNumber):
    isPrime = False
    current = 1
    for i in range (1,MaxNumber):
            isPrime = True
            for i in range(2, current):
                if (current % i == 0):
                    isPrime = False
            if(isPrime == False):
                current+=1
            else:
                yield current
                current+=1
```

```
class primeNumbers():
    def __init__(self,maxNumber):
        self.max = maxNumber
        self.current=0
    def __iter__(self):
        return self
    def __next__(self):
        isPrime = False
        while (isPrime== False & self.current<=self.max):
            self.current += 1
            isPrime = True
            for i in range(2,self.current):
                if(self.current%i == 0 ):
                    isPrime = False
        return self.current
```

```
A= primeNumbers(10)
A.__iter__()
print(A.__next__())
print(A.__next__())
print(A.__next__())
```