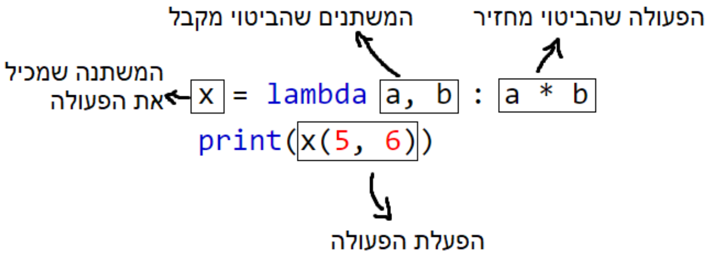
Functional

imperative - using loops and such, explicit change of states

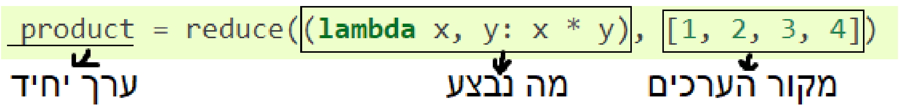
functional - composing functions on one another, each behaving "like a "black box

Lambda



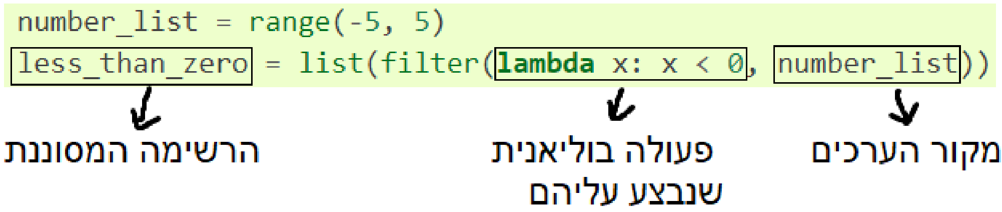
```
x = lambda a, b : a * b
print(x(5, 6))
```

Reduce- ערך יחיד



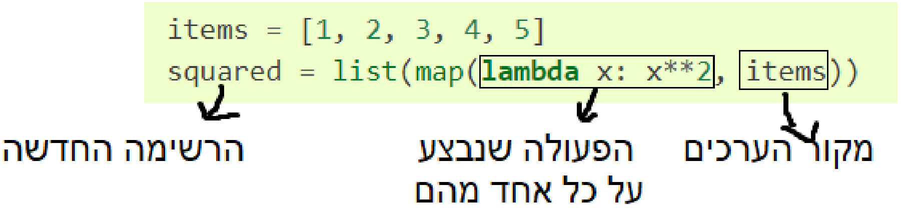
```
product = reduce((lambda x, y: x * y), [1, 2, 3, 4])
```

Filter- גודל קטן או שווה לרשימה



```
number_list = range(-5, 5)
less_than_zero = list(filter(lambda x: x < 0, number_list))
```

Map- גודל שווה לרשימה



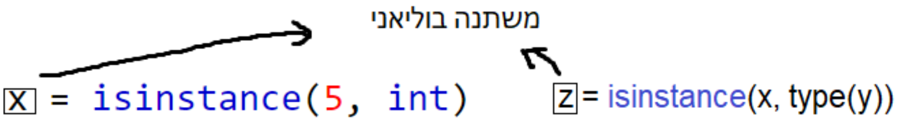
```
items = [1, 2, 3, 4, 5]
squared = list(map(lambda x: x**2, items))
```

Any\All

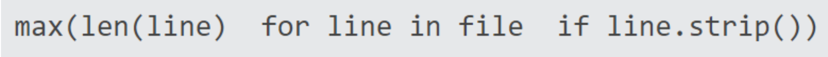
`any([False, True, False, False])` מחזיר אם קיים ערך אחד ברשימה שהוא True

`all([False, False, False])` מחזיר האם כל הערעים ברשימה הם True

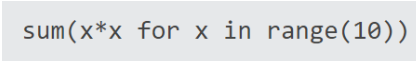
isinstance



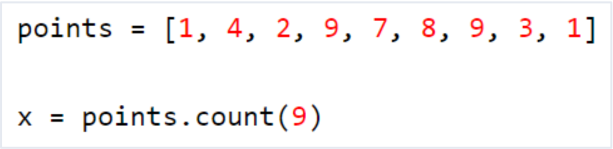
```
x = isinstance(5, int)
z = isinstance(x, type(y))
```



```
max(len(line)  for line in file  if line.strip())
```



```
sum(x*x for x in range(10))
```



```
points = [1, 4, 2, 9, 7, 8, 9, 3, 1]

x = points.count(9)
```

Data structures

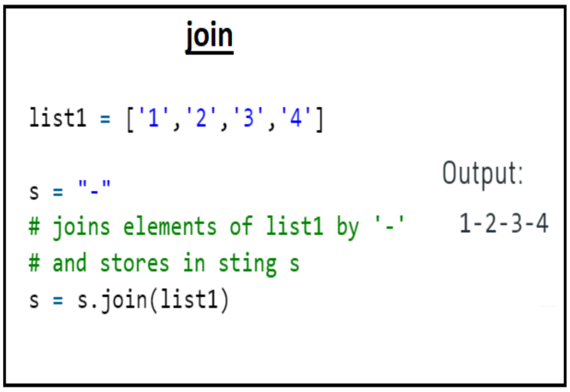
Set****	Tuple	List	String	To / From

set(myStr) *	tuple(myStr) *	list(myStr)*
		myStr.split(sep)***
				String

set(myList)	tuple(myList)		sep.join(myList)**	List
set(myTuple)		list(myTuple)	sep.join(myTuple)**	Tuple
	tuple(mySet)	list(mySet)	sep.join(mySet)**	Set****

* splits by char
** joins by sep
*** splits by sep

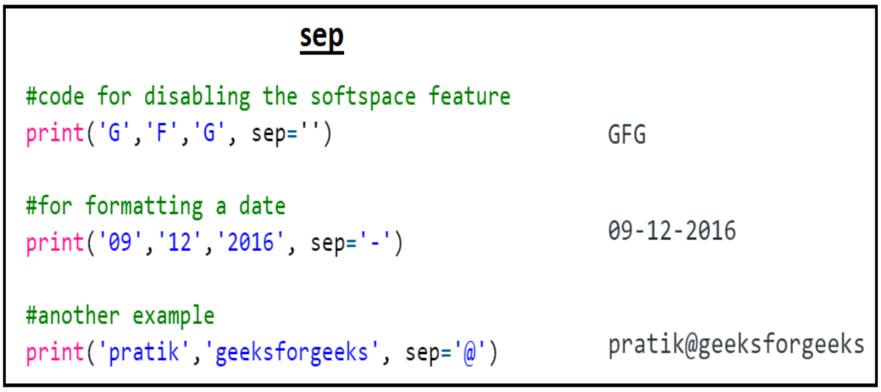
String



join

```
list1 = ['1','2','3','4']

s = "-"
# joins elements of list1 by '-'
# and stores in sting s
s = s.join(list1)
```

```
Output:
1-2-3-4
```

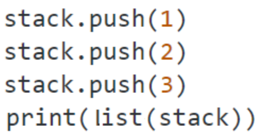


sep

```
#code for disabling the softspace feature
print('G','F','G', sep='')                          GFG

#for formatting a date
print('09','12','2016', sep='-')                    09-12-2016

#another example
print('pratik','geeksforgeeks', sep='@')            pratik@geeksforgeeks
```

Stack/Deque

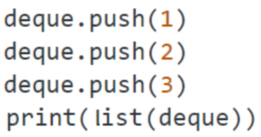
use append not push



```
stack.push(1)
stack.push(2)
stack.push(3)
print(list(stack))
```

```
// prints 1, 2, 3
```

סדר לפי הכנסה למחסנית

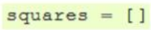


```
deque.push(1)
deque.push(2)
deque.push(3)
print(list(deque))
```

```
// prints 3, 2, 1
```

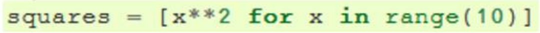
סדר מיקום במחסנית

List[]



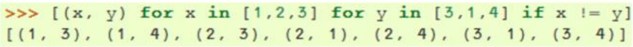
```
squares = []
```

יצירת רשימה ריקה



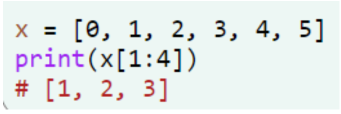
```
squares = [x**2 for x in range(10)]
```

יצירת רשימה מספרית

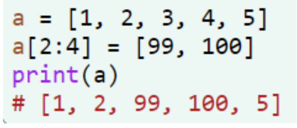


```
>>> [(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

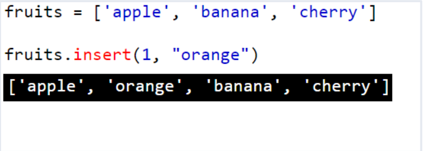
יצירת רשימה מרשימות



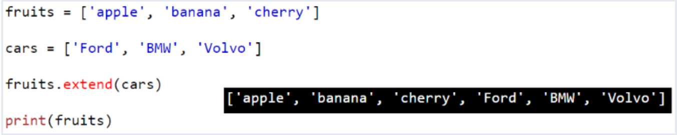
```
x = [0, 1, 2, 3, 4, 5]
print(x[1:4])
# [1, 2, 3]
```



```
a = [1, 2, 3, 4, 5]
a[2:4] = [99, 100]
print(a)
# [1, 2, 99, 100, 5]
```



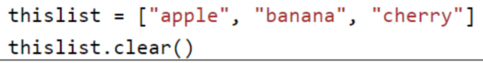
```
fruits = ['apple', 'banana', 'cherry']

fruits.insert(1, "orange")
['apple', 'orange', 'banana', 'cherry']
```



```
fruits = ['apple', 'banana', 'cherry']

cars = ['Ford', 'BMW', 'Volvo']

fruits.extend(cars)
                         ['apple', 'banana', 'cherry', 'Ford', 'BMW', 'Volvo']
print(fruits)
```

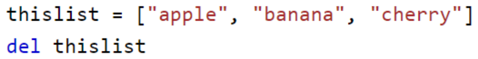
Remove\Delete

ניקוי רשימה



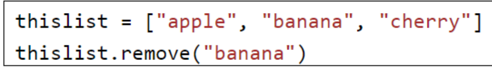
```
thislist = ["apple", "banana", "cherry"]
thislist.clear()
```

מחיקת רשימה



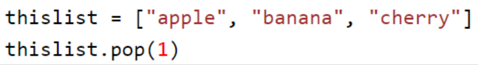
```
thislist = ["apple", "banana", "cherry"]
del thislist
```

הסרה לפי׳ שם



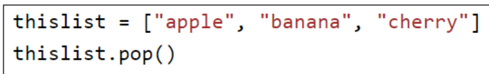
```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
```

הסרה לפי מיקום



```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
```

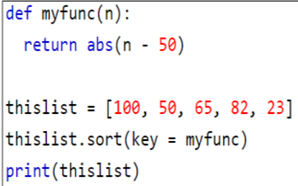
הסרת האיבר האחרון



```
thislist = ["apple", "banana", "cherry"]
thislist.pop()
```

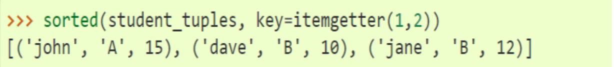
Sort

מיון ע"י פונקציה

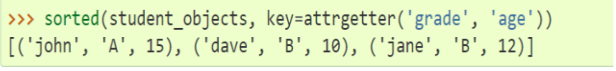


```
def myfunc(n):
  return abs(n - 50)

thislist = [100, 50, 65, 82, 23]
thislist.sort(key = myfunc)
print(thislist)
```

מיון לפי מס' פרמטרים (ראשי ומשני)



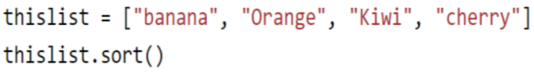
```
>>> sorted(student_tuples, key=itemgetter(1,2))
[('john', 'A', 15), ('dave', 'B', 10), ('jane', 'B', 12)]
```



```
>>> sorted(student_objects, key=attrgetter('grade', 'age'))
[('john', 'A', 15), ('dave', 'B', 10), ('jane', 'B', 12)]
```

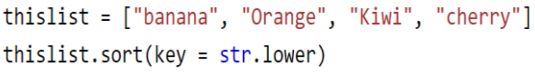
מיון אלפבתי

עולה



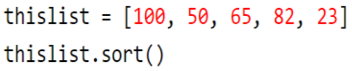
```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.sort()
```

יורד



```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.sort(key = str.lower)
```

מיון לפי ערך מספרי

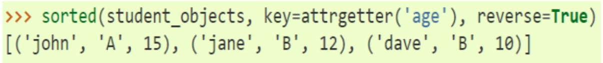


```
thislist = [100, 50, 65, 82, 23]
thislist.sort()
```

בחירת משתנה למיון

```
data = [("Apples", 5, "20"), ("Pears", 1, "5"), ("Oranges", 6, "10")]
data.sort(key=lambda x:x[1])
#output:  [('Pears', 1, '5'), ('Apples', 5, '20'), ('Oranges', 6, '10')]
```

מיון בסדר הפוך



```
>>> sorted(student_objects, key=attrgetter('age'), reverse=True)
[('john', 'A', 15), ('jane', 'B', 12), ('dave', 'B', 10)]
```

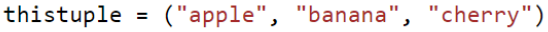
Set{}

רשימה ללא כפילויות

Tuple()

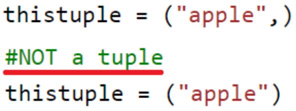
רצף לא ניתן לשינוי לאחר יצירה

יותר מאיבר אחד



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

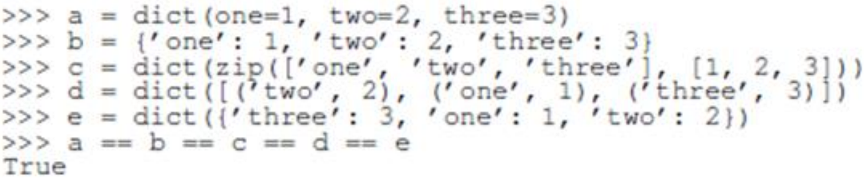
איבר אחד



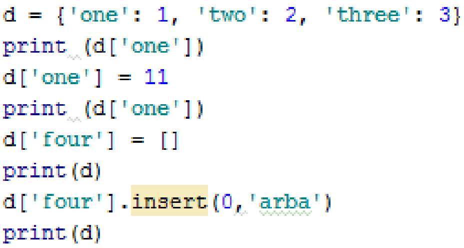
```
thistuple = ("apple",)
#NOT a tuple
thistuple = ("apple")
```

Dictionaries{}

{ key : value }

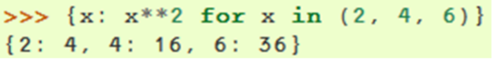


```
>>> a = dict(one=1, two=2, three=3)
>>> b = {'one': 1, 'two': 2, 'three': 3}
>>> c = dict(zip(['one', 'two', 'three'], [1, 2, 3]))
>>> d = dict([('two', 2), ('one', 1), ('three', 3)])
>>> e = dict({'three': 3, 'one': 1, 'two': 2})
>>> a == b == c == d == e
True
```



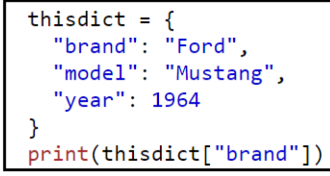
```
d = {'one': 1, 'two': 2, 'three': 3}
print (d['one'])
d['one'] = 11
print (d['one'])
d['four'] = []
print(d)
d['four'].insert(0,'arba')
print(d)
```

יצירת dict באמצעות for



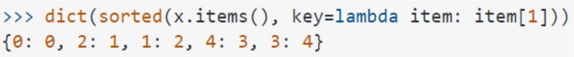
```
>>> {x: x**2 for x in (2, 4, 6)}
{2: 4, 4: 16, 6: 36}
```

מציאת ערך ע"י KEY



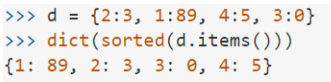
```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict["brand"])
```

מיון לפי ערך

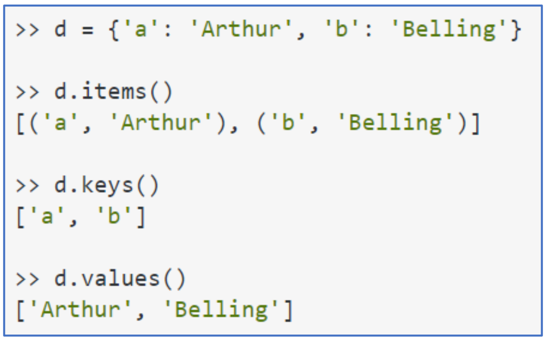


```
>>> dict(sorted(x.items(), key=lambda item: item[1]))
{0: 0, 2: 1, 1: 2, 4: 3, 3: 4}
```

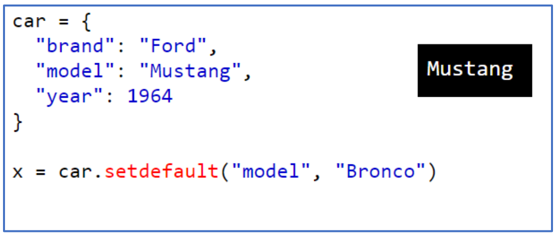
מיון לפי key



```
>>> d = {2:3, 1:89, 4:5, 3:0}
>>> dict(sorted(d.items()))
{1: 89, 2: 3, 3: 0, 4: 5}
```

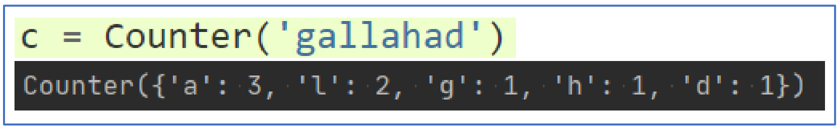


```
>> d = {'a': 'Arthur', 'b': 'Belling'}

>> d.items()
[('a', 'Arthur'), ('b', 'Belling')]

>> d.keys()
['a', 'b']

>> d.values()
['Arthur', 'Belling']
```

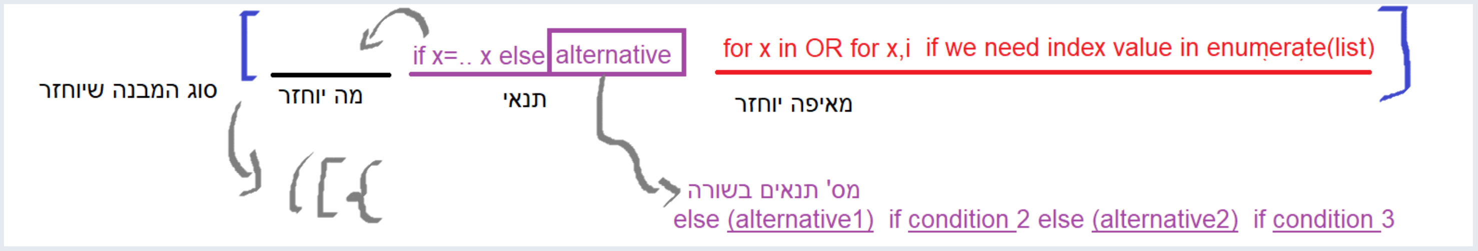


```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

x = car.setdefault("model", "Bronco")
```

```
Mustang
```



```
c = Counter('gallahad')
Counter({'a': 3, 'l': 2, 'g': 1, 'h': 1, 'd': 1})
```



[_____ if x=.. x else alternative   for x in OR for x,i if we need index value in enumerate(list)]

סוג המבנה שיחזור   מה יחזור   תנאי   מאיפה יחזור

([{

מס' תנאים בשורה

else (alternative1) if condition 2 else (alternative2) if condition 3

Generator/Iterator/Generator expression

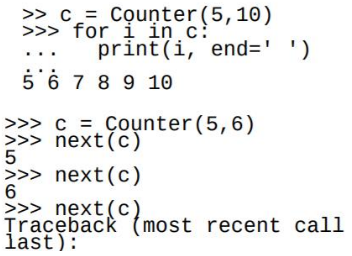
Iterator

```
class Counter(object):
    def __init__(self, low, high):
        self.current = low
        self.high = high

    def __iter__(self):
        return self

    def __next__(self):
        if self.current > self.high:
            raise StopIteration
        else:
            self.current += 1
            return self.current - 1
```

low- ערך ההתחלה
high- ערך מקסימלי

'Returns itself as an iterator object'

'Returns the next value till current is lower than high'



```
>> c = Counter(5,10)
>>> for i in c:
...     print(i, end=' ')
...
5 6 7 8 9 10

>>> c = Counter(5,6)
>>> next(c)
5
>>> next(c)
6
>>> next(c)
Traceback (most recent call
last):
```

Generator

```
def numberGenerator(n):
    number = 0
    while number < n:
        yield number
        number += 1

myGenerator = numberGenerator(3)
```

output

```
0
1
2
```

```
def numberGenerator(n):
  if n < 20:
     number = 0
     while number < n:
        yield number
        number += 1
  else:
     return

print(list(numberGenerator(30)))
```

return בתנאי עצירה

```
g = numberGenerator(10)

counter = 0

while counter < 10:
   print(next(g))
   counter += 1
```

שימוש בnext
על גנרטור

פלט:

```
0
1
2
3
4
5
6
7
8
9
```

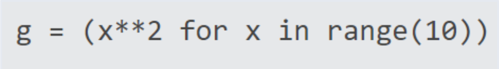
```
def numberGenerator(n):
     number = yield
     while number < n:
         number = yield number
         number += 1

g = numberGenerator(10)    # Create our generator
next(g)                    #
print(g.send(5))
```

מעדכן את number
ולאחר מכן מחזיר אותו

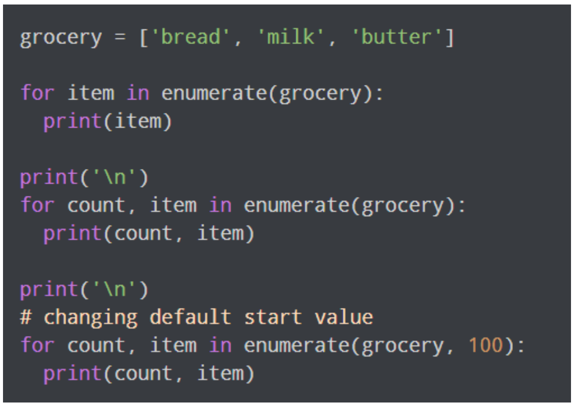
פלט: 5

Generator expression

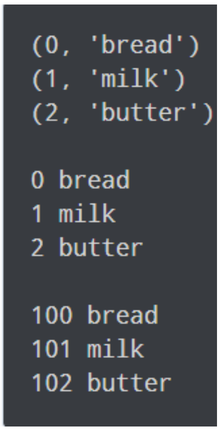
```
>>> gen_exp = (x ** 2 for x in range(10) if x % 2 == 0)
>>> for x in gen_exp:
...     print(x)
0
4
16
36
64
```



```
g = (x**2 for x in range(10))
```

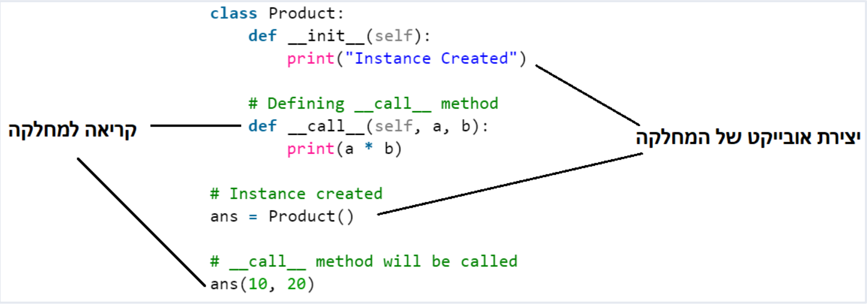


```
grocery = ['bread', 'milk', 'butter']

for item in enumerate(grocery):
  print(item)

print('\n')
for count, item in enumerate(grocery):
  print(count, item)

print('\n')
# changing default start value
for count, item in enumerate(grocery, 100):
  print(count, item)
```



```
(0, 'bread')
(1, 'milk')
(2, 'butter')

0 bread
1 milk
2 butter

100 bread
101 milk
102 butter
```

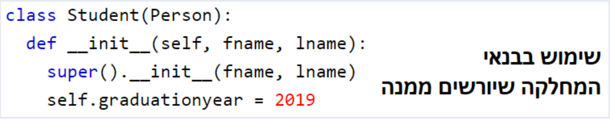
Class and Multiple inheritance



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

    # Defining __call__ method
    def __call__(self, a, b):
        print(a * b)

# Instance created
ans = Product()

# __call__ method will be called
ans(10, 20)
```

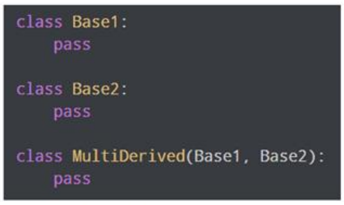
קריאה למחלקה

יצירת אובייקט של המחלקה



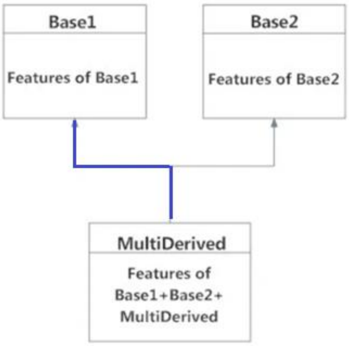
```
class Student(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)
    self.graduationyear = 2019
```

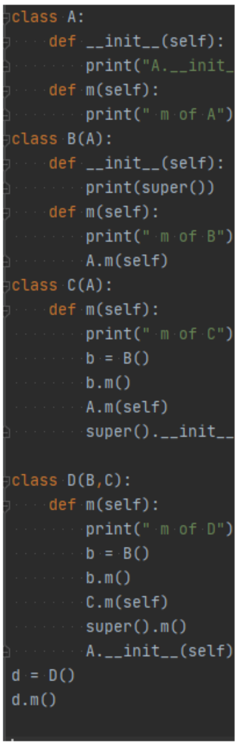
שימוש בבנאי
המחלקה שיורשים ממנה



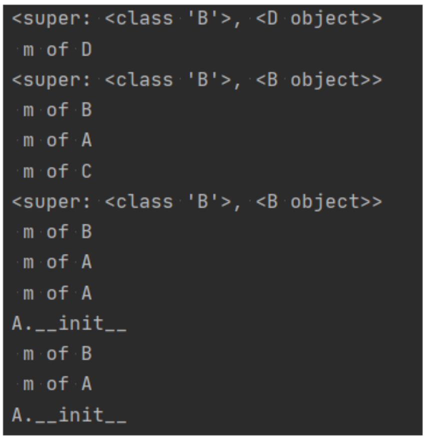
```
class Base1:
    pass

class Base2:
    pass

class MultiDerived(Base1, Base2):
    pass
```

קיימת עדיפות לפעולות של הפונקציה
שמופיעה קודם (Base1)





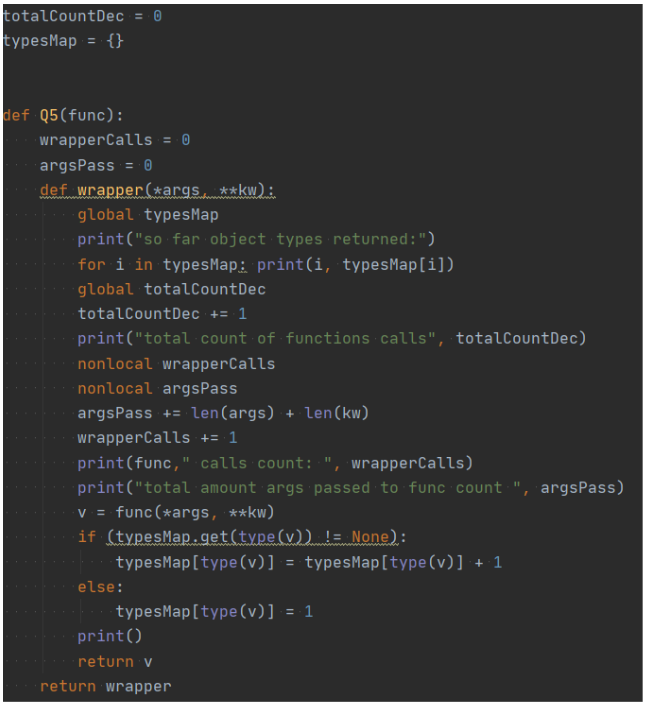
```
class A:
    def __init__(self):
        print("A.__init_
    def m(self):
        print(" m of A")
class B(A):
    def __init__(self):
        print(super())
    def m(self):
        print(" m of B")
        A.m(self)
class C(A):
    def m(self):
        print(" m of C")
        b = B()
        b.m()
        A.m(self)
        super().__init__

class D(B,C):
    def m(self):
        print(" m of D")
        b = B()
        b.m()
        C.m(self)
        super().m()
        A.__init__(self)
d = D()
d.m()
```



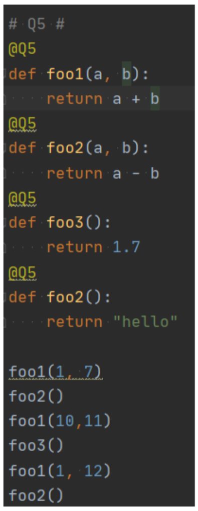
```
<super: <class 'B'>, <D object>>
 m of D
<super: <class 'B'>, <B object>>
 m of B
 m of A
 m of C
<super: <class 'B'>, <B object>>
 m of B
 m of A
 m of A
A.__init__
 m of B
 m of A
A.__init__
```

Decorator

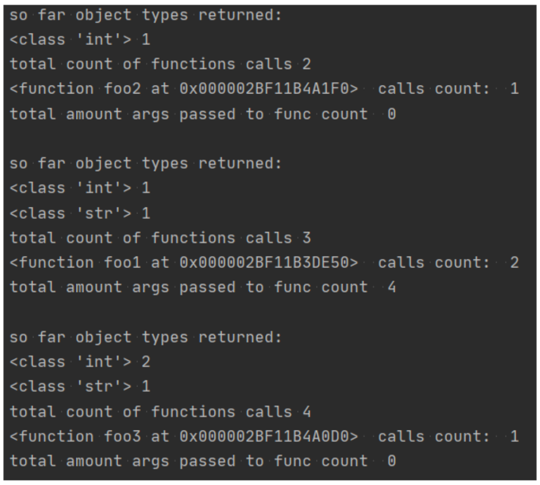
Function decorator



```python
totalCountDec = 0
typesMap = {}


def Q5(func):
    wrapperCalls = 0
    argsPass = 0
    def wrapper(*args, **kw):
        global typesMap
        print("so far object types returned:")
        for i in typesMap: print(i, typesMap[i])
        global totalCountDec
        totalCountDec += 1
        print("total count of functions calls", totalCountDec)
        nonlocal wrapperCalls
        nonlocal argsPass
        argsPass += len(args) + len(kw)
        wrapperCalls += 1
        print(func," calls count: ", wrapperCalls)
        print("total amount args passed to func count ", argsPass)
        v = func(*args, **kw)
        if (typesMap.get(type(v)) != None):
            typesMap[type(v)] = typesMap[type(v)] + 1
        else:
            typesMap[type(v)] = 1
        print()
        return v
    return wrapper
```

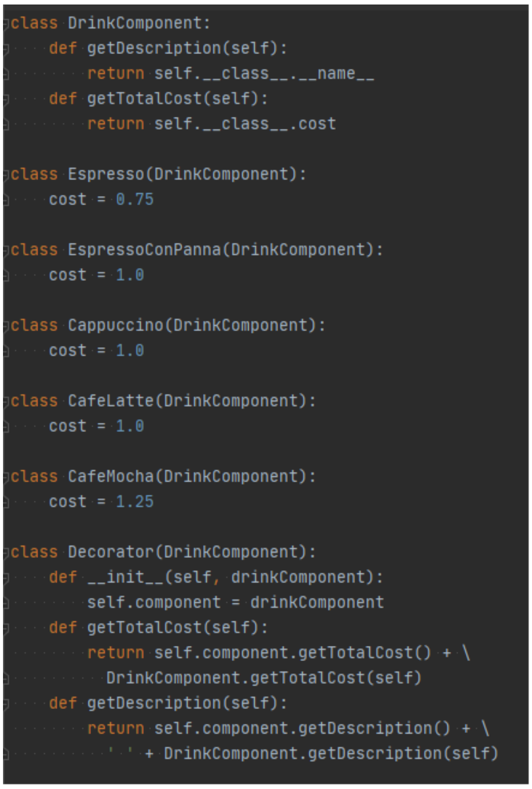


```python
# Q5 #
@Q5
def foo1(a, b):
    return a + b
@Q5
def foo2(a, b):
    return a - b
@Q5
def foo3():
    return 1.7
@Q5
def foo2():
    return "hello"

foo1(1, 7)
foo2()
foo1(10,11)
foo3()
foo1(1, 12)
foo2()
```

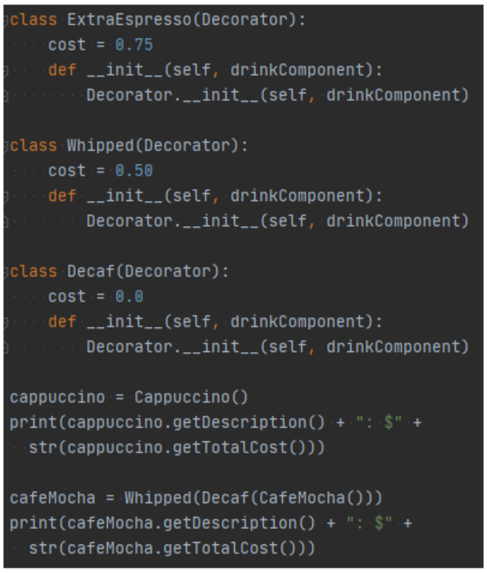


```
so far object types returned:
<class 'int'> 1
total count of functions calls 2
<function foo2 at 0x000002BF11B4A1F0>  calls count:  1
total amount args passed to func count  0

so far object types returned:
<class 'int'> 1
<class 'str'> 1
total count of functions calls 3
<function foo1 at 0x000002BF11B3DE50>  calls count:  2
total amount args passed to func count  4

so far object types returned:
<class 'int'> 2
<class 'str'> 1
total count of functions calls 4
<function foo3 at 0x000002BF11B4A0D0>  calls count:  1
total amount args passed to func count  0
```

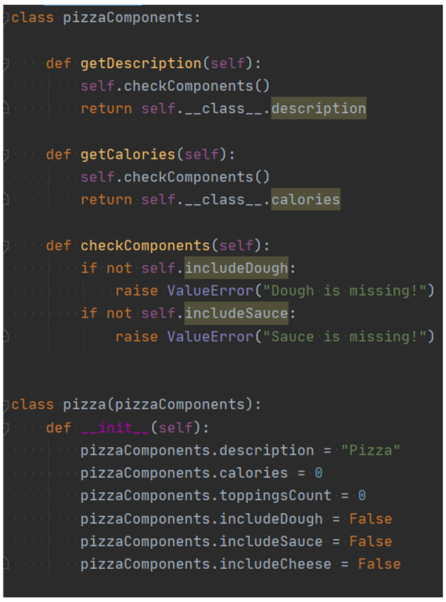
Class decorator



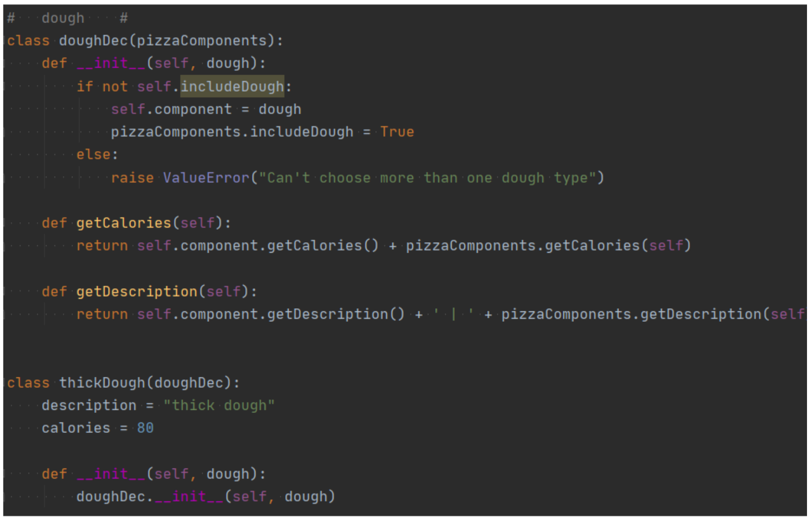
```python
class DrinkComponent:
    def getDescription(self):
        return self.__class__.__name__
    def getTotalCost(self):
        return self.__class__.cost

class Espresso(DrinkComponent):
    cost = 0.75

class EspressoConPanna(DrinkComponent):
    cost = 1.0

class Cappuccino(DrinkComponent):
    cost = 1.0

class CafeLatte(DrinkComponent):
    cost = 1.0

class CafeMocha(DrinkComponent):
    cost = 1.25

class Decorator(DrinkComponent):
    def __init__(self, drinkComponent):
        self.component = drinkComponent
    def getTotalCost(self):
        return self.component.getTotalCost() + \
          DrinkComponent.getTotalCost(self)
    def getDescription(self):
        return self.component.getDescription() + \
          ' ' + DrinkComponent.getDescription(self)
```



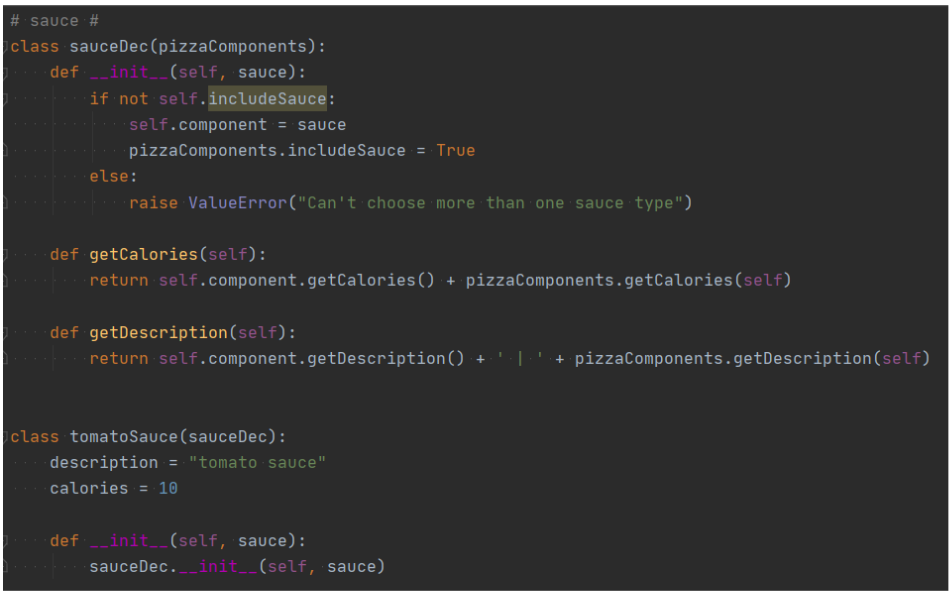
```python
class ExtraEspresso(Decorator):
    cost = 0.75
    def __init__(self, drinkComponent):
        Decorator.__init__(self, drinkComponent)

class Whipped(Decorator):
    cost = 0.50
    def __init__(self, drinkComponent):
        Decorator.__init__(self, drinkComponent)

class Decaf(Decorator):
    cost = 0.0
    def __init__(self, drinkComponent):
        Decorator.__init__(self, drinkComponent)

cappuccino = Cappuccino()
print(cappuccino.getDescription() + ": $" +
  str(cappuccino.getTotalCost()))

cafeMocha = Whipped(Decaf(CafeMocha()))
print(cafeMocha.getDescription() + ": $" +
  str(cafeMocha.getTotalCost()))
```



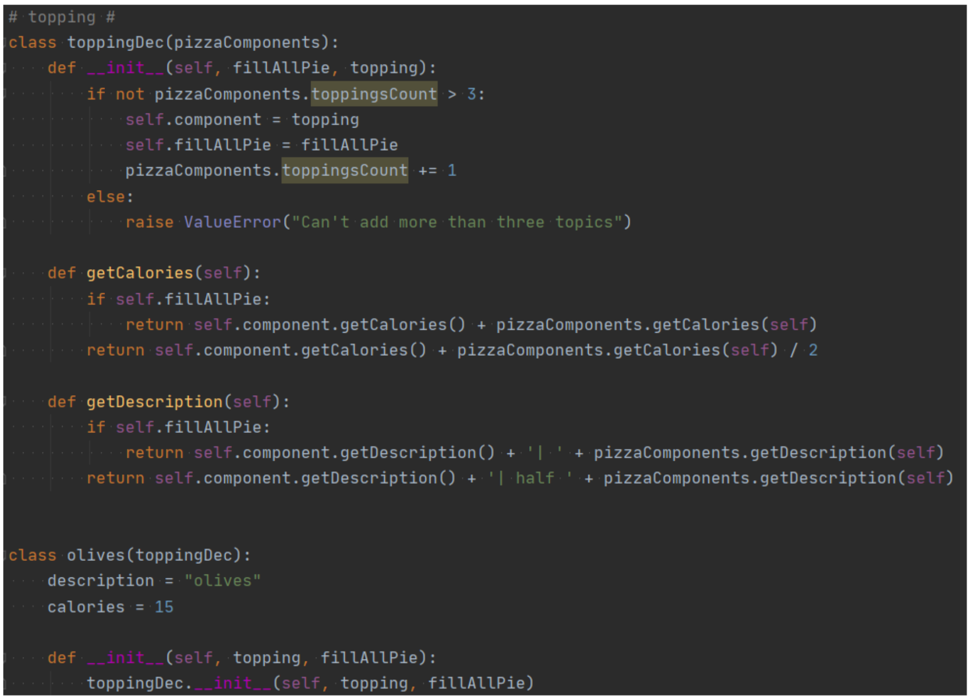
```
class pizzaComponents:

    def getDescription(self):
        self.checkComponents()
        return self.__class__.description

    def getCalories(self):
        self.checkComponents()
        return self.__class__.calories

    def checkComponents(self):
        if not self.includeDough:
            raise ValueError("Dough is missing!")
        if not self.includeSauce:
            raise ValueError("Sauce is missing!")


class pizza(pizzaComponents):
    def __init__(self):
        pizzaComponents.description = "Pizza"
        pizzaComponents.calories = 0
        pizzaComponents.toppingsCount = 0
        pizzaComponents.includeDough = False
        pizzaComponents.includeSauce = False
        pizzaComponents.includeCheese = False
```



```
#  dough   #
class doughDec(pizzaComponents):
    def __init__(self, dough):
        if not self.includeDough:
            self.component = dough
            pizzaComponents.includeDough = True
        else:
            raise ValueError("Can't choose more than one dough type")

    def getCalories(self):
        return self.component.getCalories() + pizzaComponents.getCalories(self)

    def getDescription(self):
        return self.component.getDescription() + ' | ' + pizzaComponents.getDescription(self


class thickDough(doughDec):
    description = "thick dough"
    calories = 80

    def __init__(self, dough):
        doughDec.__init__(self, dough)
```



```
# sauce #
class sauceDec(pizzaComponents):
    def __init__(self, sauce):
        if not self.includeSauce:
            self.component = sauce
            pizzaComponents.includeSauce = True
        else:
            raise ValueError("Can't choose more than one sauce type")

    def getCalories(self):
        return self.component.getCalories() + pizzaComponents.getCalories(self)

    def getDescription(self):
        return self.component.getDescription() + ' | ' + pizzaComponents.getDescription(self)


class tomatoSauce(sauceDec):
    description = "tomato sauce"
    calories = 10

    def __init__(self, sauce):
        sauceDec.__init__(self, sauce)
```

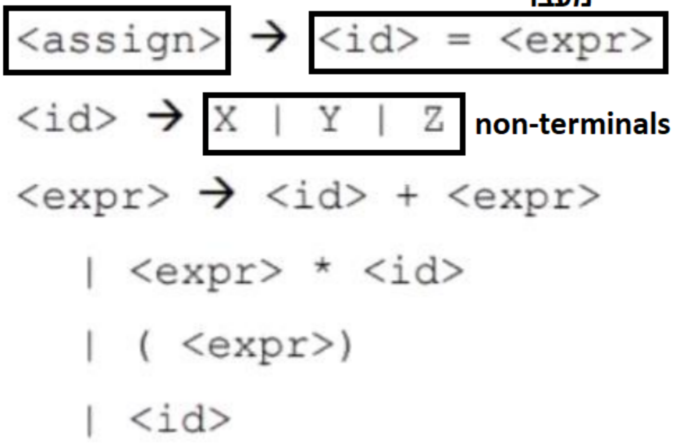


```
# topping #
class toppingDec(pizzaComponents):
    def __init__(self, fillAllPie, topping):
        if not pizzaComponents.toppingsCount > 3:
            self.component = topping
            self.fillAllPie = fillAllPie
            pizzaComponents.toppingsCount += 1
        else:
            raise ValueError("Can't add more than three topics")

    def getCalories(self):
        if self.fillAllPie:
            return self.component.getCalories() + pizzaComponents.getCalories(self)
        return self.component.getCalories() + pizzaComponents.getCalories(self) / 2

    def getDescription(self):
        if self.fillAllPie:
            return self.component.getDescription() + '| ' + pizzaComponents.getDescription(self)
        return self.component.getDescription() + '| half ' + pizzaComponents.getDescription(self)


class olives(toppingDec):
    description = "olives"
    calories = 15

    def __init__(self, topping, fillAllPie):
        toppingDec.__init__(self, topping, fillAllPie)
```

BNF

טרמינל ההתחלה

מעבר

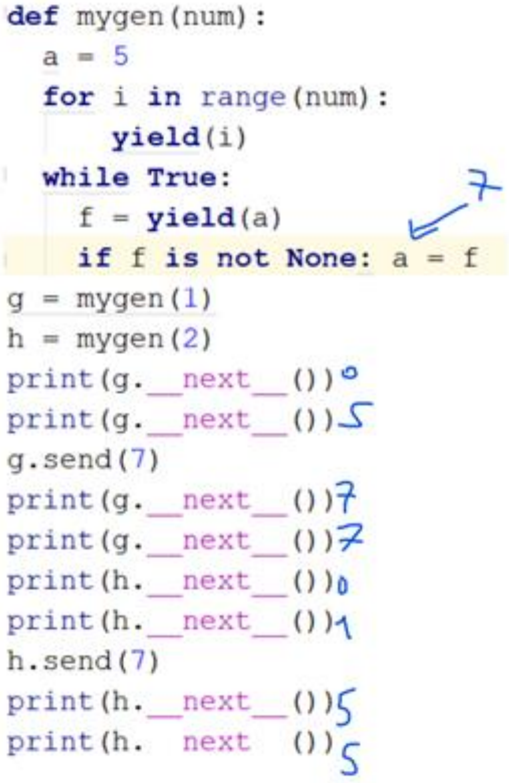


```
<assign> → <id> = <expr>
<id> → X | Y | Z
<expr> → <id> + <expr>
    | <expr> * <id>
    | ( <expr>)
    | <id>
```

non-terminals

הביטוי הסופי חייב להכיל
רק non-terminals

תוספת דוגמה לגנרטור



```
def mygen(num):
    a = 5
    for i in range(num):
        yield(i)
    while True:
        f = yield(a)          7
        if f is not None: a = f
g = mygen(1)
h = mygen(2)
print(g.__next__())0
print(g.__next__())5
g.send(7)
print(g.__next__())7
print(g.__next__())7
print(h.__next__())0
print(h.__next__())1
h.send(7)
print(h.__next__())5
print(h.__next__())5
```