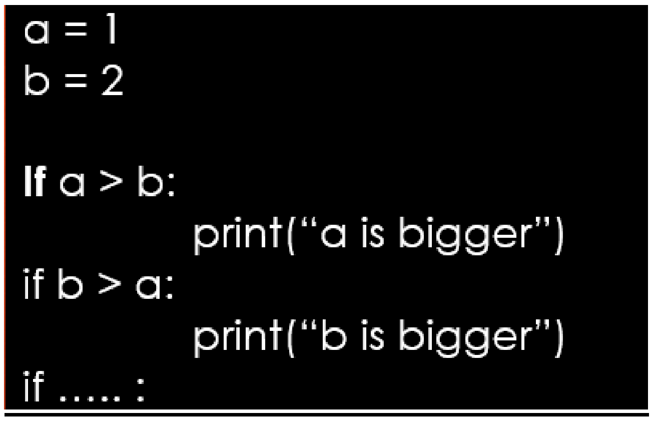
סיכום שיעור שני:

1. **תנאים:**

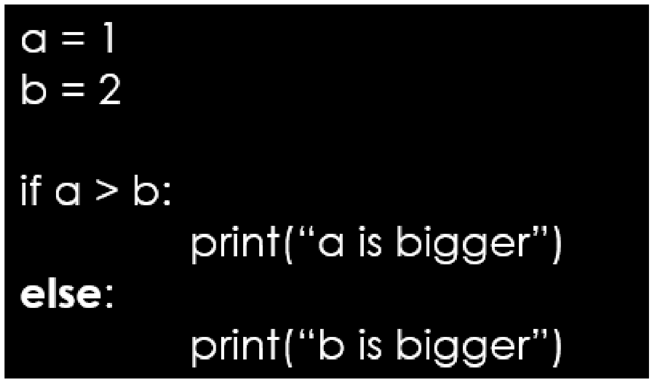
תנאי הוא ביטוי בוליאני, שמחזיר True / False . אם התנאי נכון, אז ה statement מבוצע
https://docs.python.org/3.6/reference/compound_stmts.html

- If - תנאי פשוט



```
a = 1
b = 2

If a > b:
          print("a is bigger")
if b > a:
          print("b is bigger")
if ..... :
```

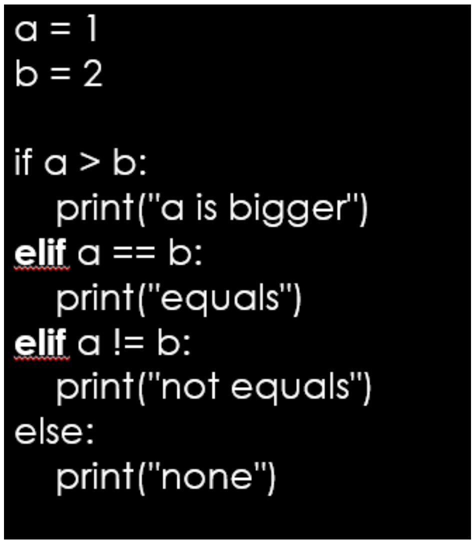
- else- ייכתב בצמוד ל if ויתבצע במידה והתנאי ב if אינו מתקיים.



```
a = 1
b = 2

if a > b:
          print("a is bigger")
else:
          print("b is bigger")
```

- elif- מיועד ליצירת מערכת תנאים

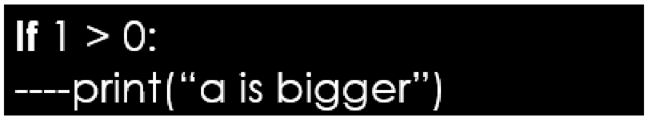
- The elif statement allows you to check multiple expressions for TRUE and execute a block of code as soon as one of the conditions evaluates to TRUE.
- Similar to the else, the elif statement is optional.
- Unlike else, there can be a few of elif statements following an if.
- As soon as one condition is met, the rest will not be tested.



```
a = 1
b = 2

if a > b:
   print("a is bigger")
elif a == b:
   print("equals")
elif a != b:
   print("not equals")
else:
   print("none")
```

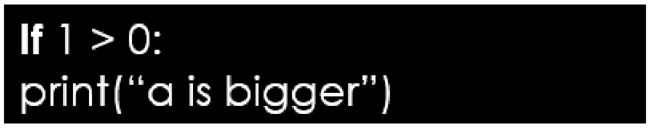
2. Indentation:

- Unlike many other languages, where each block of code is marked using brackets { } In Python, each scope (block of code), is marked by using indentation.
- Indentation basically means spaces.
- Each Indentation should be multiple of four (4/8/12/…)
- **At the below code, we have 4 spaces before print, to associate it with the if statement above**



```
If 1 > 0:
----print("a is bigger")
```

- No spaces - will cause an error



```
If 1 > 0:
print("a is bigger")
```

:Statements .3

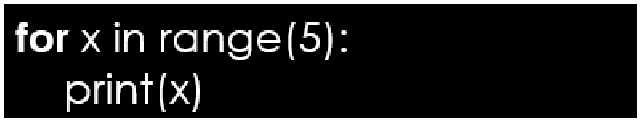
https://docs.python.org/3.6/reference/compound_stmts.html

Loops:

- A loop is a sequence of statements which is specified once but which may be carried out several times in succession.
- The code "inside" the loop (the *body* of the loop) is obeyed a specified number of times, or once for each of a collection of items, or until some condition is met, or indefinitely.
- In Python we have a few loops types:
  - For
  - While

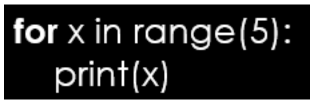
For loop:

- Use for loop when the number of the iterations are known before entering into the body of the loop.
- It has flexibility to assign variables before entering into the body of the loop and perform an update at the end of the loop.
- Syntax contain 3 parameters:
  - Variable initialization (x).
  - Condition (range 5).
  - Statement (printing).
- To put the code above in simple words will be: ***Increment X value by 1 in each iteration, and keep running as long as X is smaller than 5***

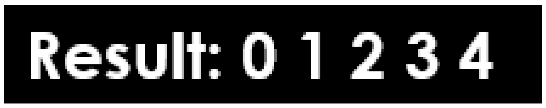


```
for x in range(5):
    print(x)
```

- For loop has a few variations:
  - Run for X times:

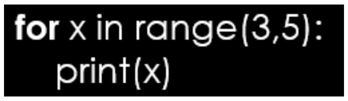


```
for x in range(5):
    print(x)
```

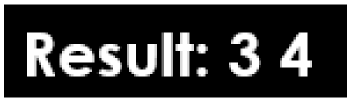


```
Result: 0 1 2 3 4
```

  - Run from a specific index(3) X times(5):

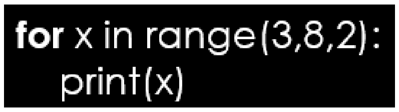


```
for x in range(3,5):
    print(x)
```

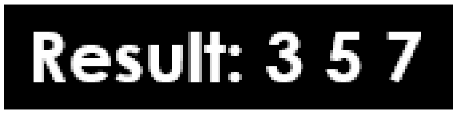


```
Result: 3 4
```

  - Run from a specific index X times, but increment X in 2 every iteration:



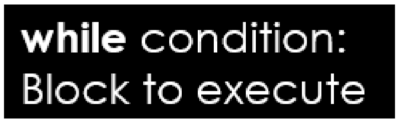
```
for x in range(3,8,2):
    print(x)
```



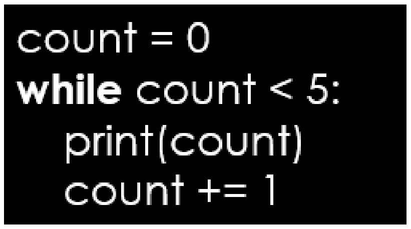
```
Result: 3 5 7
```

- While loop:

- The block of statements will be repeated as long as the condition returns true.
- The condition should return false for exiting the loop.
- The ‘while’ loop can be used when the number of iterations is unknown.

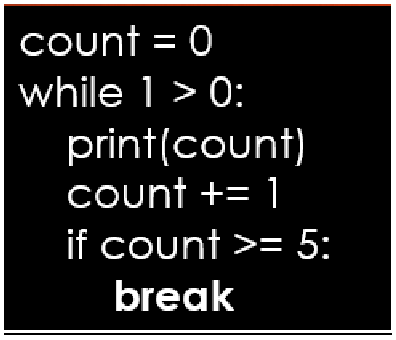


```
while condition:
Block to execute
```



```
count = 0
while count < 5:
    print(count)
    count += 1
```

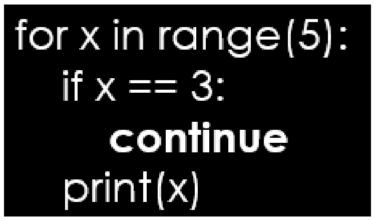
- Break statement
  When a break statement is encountered inside a loop, the loop is terminated and program control resumes at the next statement following the loop.



```
count = 0
while 1 > 0:
    print(count)
    count += 1
    if count >= 5:
        break
```

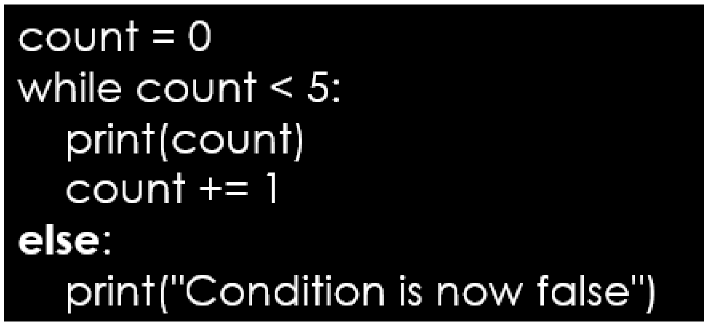
- Continue statement

  Continue statement forces an early iteration of the loop



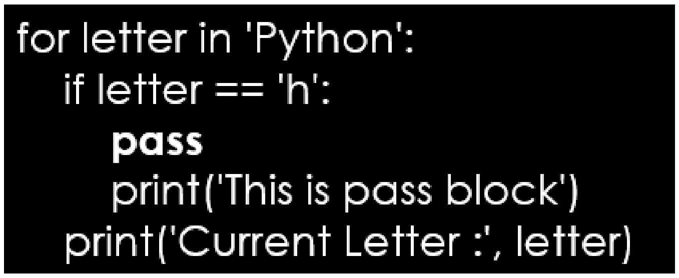
```
for x in range(5):
    if x == 3:
        continue
    print(x)
```

- <u>Else in a loop</u>
  Python supports to have an else statement associated with a loop statements.
  - If the **else** statement is used with a for loop, the else statement is executed when the loop has exhausted iterating the list.
  - If the **else** statement is used with a while loop, the else statement is executed when the condition becomes false.



```
count = 0
while count < 5:
   print(count)
   count += 1
else:
   print("Condition is now false")
```

- <u>Pass statement</u>
- When an external condition is triggered, the **pass** statement allows you to handle the condition without the loop being impacted in any way.
- All of the code will continue to be read unless a break or other statement occurs.
- As with the other statements, the **pass** statement will be within the block of code under the loop statement, typically after a conditional if statement.

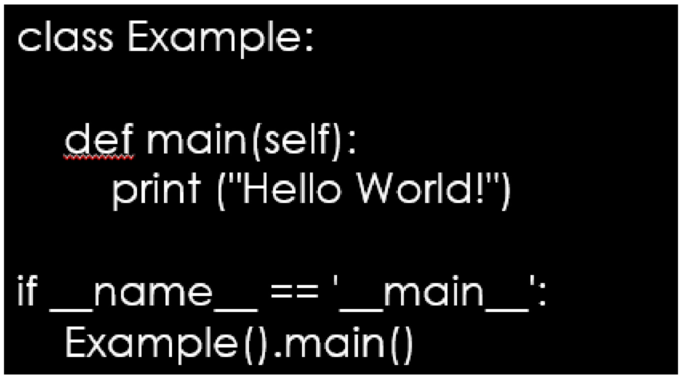


```
for letter in 'Python':
   if letter == 'h':
      pass
      print('This is pass block')
   print('Current Letter :', letter)
```

Modules (files) & Classes .4

- In Python, one file is called a module.
- A module can consist of multiple classes or functions.
- As Python is not an OO language only, it does not make sense to have a rule that says, one file should only contain one class.
- One file (module) should contain classes / functions that belong together, i.e. provide similar functionality or depend on each other.
- Earlier, we wrote our code in a file (module) without using classes / functions.

- In order to use classes, we will need to add a few things to our module:
  - Start our code with the word class, writing the word class with any word to define the class.
  - Create a main function, add you logic into the function (for example; print), we are using the words def & self, which will be explained later.
  - Create an entry point to our program which will call our main function

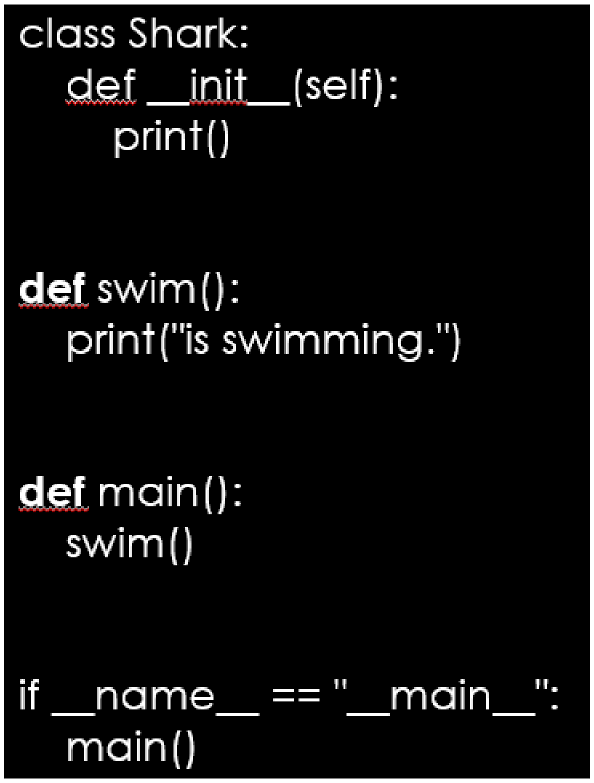


```
class Example:

    def main(self):
        print ("Hello World!")

if __name__ == '__main__':
    Example().main()
```

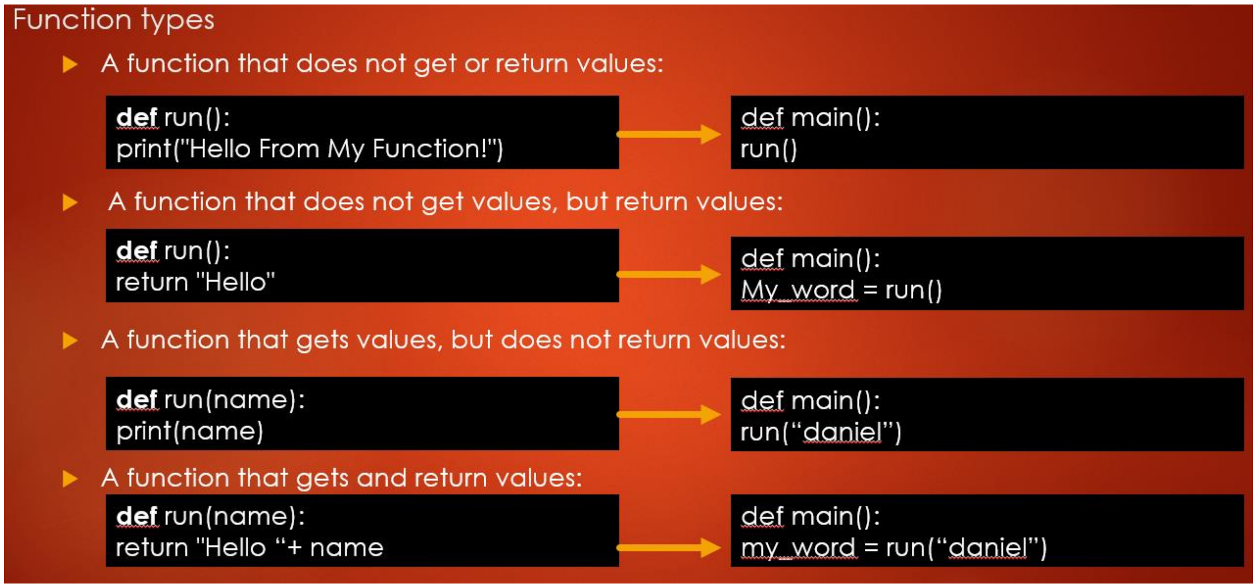
Functions .5

https://anh.cs.luc.edu/python/hands-on/3.1/handsonHtml/functions.html

- Functions are a convenient way to divide your code into useful blocks.
- A function usually consists of a sequence of
- statements to perform an action, and possibly an output value (called the return value) of some kind.
- Functions in python are defined using the keyword "def", followed with the function's name
- Main advantages:
  - Code reusability.
  - Code optimization.
  - Readable code



```
class Shark:
    def __init__(self):
        print()


def swim():
    print("is swimming.")


def main():
    swim()


if __name__ == "__main__":
    main()
```



Function types

- A function that does not get or return values:

```
def run():
print("Hello From My Function!")
```

```
def main():
run()
```

- A function that does not get values, but return values:

```
def run():
return "Hello"
```

```
def main():
My_word = run()
```

- A function that gets values, but does not return values:

```
def run(name):
print(name)
```

```
def main():
run("daniel")
```

- A function that gets and return values:

```
def run(name):
return "Hello "+ name
```

```
def main():
my_word = run("daniel")
```

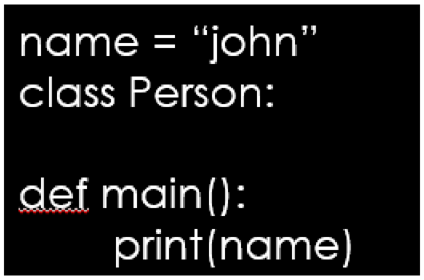
Variables .6

Python Programming language defines 2 kinds of variables:

a. Global variables

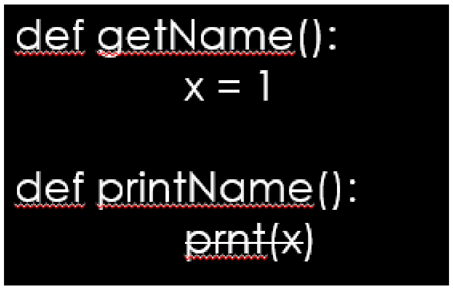
b. Local Variables

- Global variables (known also as Instance variables) are variables that are declare inside a class but outside any function, constructor or block or property.
- The idea is a variable that can be accessible anywhere in the class.
- In the following example name is an instance variable of Person class.



```
name = “john”
class Person:

def main():
      print(name)
```

- Local variables are declared in functions, constructor or blocks.
- Local variables are initialized when function or constructor block start and will be destroyed once its end.
- The variable will only be accessible from within the function.
- At the example below x is a local variable, and therefore printName can’t use it.



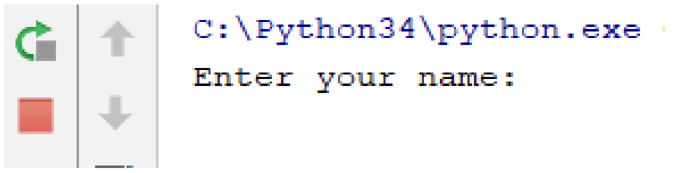
```
def getName():
          x = 1

def printName():
          ~~prnt(x~~)
```

<u>Input</u> .7

- There are hardly any programs without any input.
- Input can come in various ways, for example from a database, server and files.
- Python provides the function input(). input has an optional parameter, which is the prompt string.
- If the input function is called, the program flow will be stopped until the user has given an input and has ended the input with the return (enter) key.
- The input of the user will be interpreted, for example if the user type in an integer value, the input function returns this integer value.
- Let’s see:

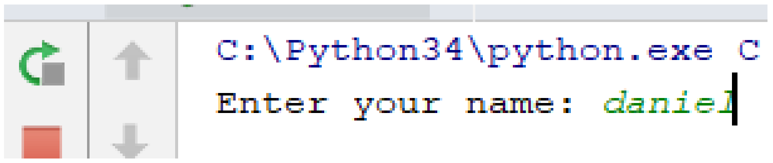
```
name = input("Enter your name: ")
print("Your name is:", name)
```

- When you will run your program, you will see the following inside the console:



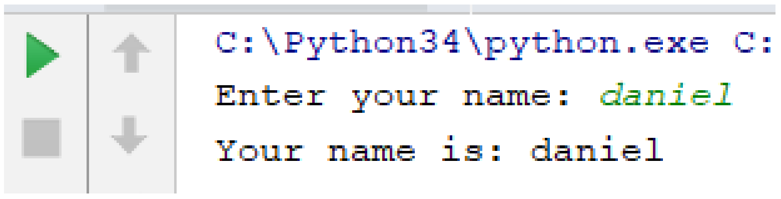
```
C:\Python34\python.exe
Enter your name:
```

- Press with the mouse underneath, and type your name:



```
C:\Python34\python.exe C
Enter your name: daniel
```

- Once your done press return (enter), and you will see the output:



```
C:\Python34\python.exe C:
Enter your name: daniel
Your name is: daniel
```

<u>תזכורת ממה שעשינו בכיתה</u>

```python
# if statement
if 1 > 0:
    print(True)

# if-else statement
if 1 > 2:
    print(True)
else:
    print(False)
# if-elif-else statement
if 1 > 2:
    print(True)
elif 1==2:
    print("wrong")
else:
    print(False)
# for loop
for x in range(10):
    print(x)

# while loop
while count < 5:
    print(count)
    count += 1

# break statement
count = 0
while 1 > 0:
    print(count)
    count += 1
    if count >= 5:
        break

# continue statement
for x in range(5):
    if x == 3:
        continue
    print(x)

# pass statement
for letter in 'Python':
    if letter == 'h':
        pass
        print('This is pass block')
    print('Current Letter :', letter)


# create a class
class Example:

#function example
    def main(self):
        print ("Hello World!")

#create entry point
if __name__ == '__main__':
    Example().main()
```