University of Tripoli
Faculty of Information Technology
Department of Software Engineering

**ITSE305 مواضيع مختارة**
**Python Programming**
**S2025**

Lecture (7): Built-in Functions

## Python Built-in Functions

- all() Function
- any() Function
- enumerate() Function
- filter() Function
- help() Function
- map() Function
- max() Function
- min() Function
- print() Function
- range() Function
- reversed() Function
- sorted() Function
- sum() Function
- zip() Function

## Python all() Function

- The all() function returns True if all items in an iterable are true, otherwise it returns False.
- If the iterable object is empty, the all() function also returns True.
- When used on a dictionary, the all() function checks if all the *keys* are true, not the *values*.
- Syntax

all(*iterable*)

Where iterable is list, tuple or dictionary



## Python all() Function - Examples

```
myNumlist = [19, 5, 8 , 0]
x = all(myNumlist)
print(x)

myTextlist = ["Ahmed", "omar", ""]
x = all(myTextlist)
print(x)

mytuple = (0, True, False)
x = all(mytuple)
print(x)

myset = {0, 1, 0}
x = all(myset)
print(x)

mydict = {0 : "Apple", 1 : "Orange"}
x = all(mydict)
print(x)
```

```
False
False
False
False
False
```

```
myNumlist = [19, 5, 8 , 1]
x = all(myNumlist)
print(x)

myTextlist = ["Ahmed", "omar", "Ali"]
x = all(myTextlist)
print(x)

mytuple = (1, True, True)
x = all(mytuple)
print(x)

myset = {1, 1, True}
x = all(myset)
print(x)

mydict = {1 : "Apple", 2 : "Orange"}
x = all(mydict)
print(x)
```

```
True
True
True
True
True
```

## Python any() Function

- The any() function returns True if any item in an iterable are true, otherwise it returns False.
- If the iterable object is empty, the any() function will return False.
- When used on a dictionary, the any() function checks if any of the *keys* are true, not the *values*.
- Syntax

  any(*iterable*)



## Python any() Function - Examples

```
myNumList = [12, 0, 17, 30, 0]
x = any(myNumList)
print(x)

myTextList = ["Omar", "Ahmed", ""]
x = any(myTextList)
print(x)

mylist = [False, True, False]
x = any(mylist)
print(x)

mytuple = (0, 1, False)
x = any(mytuple)
print(x)

myset = {0, 1, 0}
x = any(myset)
print(x)

mydict = {0 : "Apple", 1 : "Orange"}
x = any(mydict)
print(x)
```

```
True
True
True
True
True
True
```

## Python enumerate() Function

- The enumerate() function takes a collection (e.g. a tuple) and returns it as an enumerate object.
- The enumerate() function adds a counter as the key of the enumerate object.
- Syntax

  enumerate(*iterable*, *start*)

where start is a number, defining the start number of the enumerate object (the default value is 0)



## Python enumerate() Function - Examples

```
x = ('apple', 'banana', 'cherry')
y = enumerate(x)

print(list(y))
```

```
[(0, 'apple'), (1, 'banana'), (2, 'cherry')]
```

```
a = ["Examples", "for", "Enumerate"]

for i, name in enumerate(a):
    print(f"Index {i}: {name}")

print(list(enumerate(a)))
```

```
Index 0: Examples
Index 1: for
Index 2: Enumerate
[(0, 'Examples'), (1, 'for'), (2, 'Enumerate')]
```

## Python filter() Function

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

  filter(*function*, *iterable*)

Where function is that to be run for each item in the iterable



## Python filter() Function - Examples

```
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)
```

```
18
24
32
```

```
def even(n):
    return n % 2 == 0

a = [1, 2, 3, 4, 5, 6]
b = filter(even, a)

print(list(b))
```

```
[2, 4, 6]
```

## Python help() Function

- It is useful for retrieving information on various Python objects.

```
help(print)
```

```
Help on built-in function print in module builtins:

print(*args, sep=' ', end='\n', file=None, flush=False)
    Prints the values to a stream, or to sys.stdout by default.

    sep
      string inserted between values, default a space.
    end
      string appended after the last value, default a newline.
    file
      a file-like object (stream); defaults to the current sys.stdout.
    flush
      whether to forcibly flush the stream.
```



## Python map() Function

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

  map(*function*, *iterables*)

## Python map() Function - Examples

```
def myfunc(a):
  return len(a)

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

print(list(x))

################################################

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

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

print(list(x))
```

```
[5, 6, 6]
['appleorange', 'bananalemon', 'cherrypineapple']
```

```
s = ['1', '2', '3', '4']
res = map(int, s)
print(list(res))
```

```
[1, 2, 3, 4]
```



## Python max() Function

- The max() function returns the item with the highest value, or the item with the highest value in an iterable.
- If the values are strings, an alphabetically comparison is done.
- Syntax

max(*n1, n2, n3, ...*)

Or

max(*iterable*)

## Python max() Function- Examples

```
x = max(5, 10)
print(x)

##################

x = max("Mike", "John", "Vicky")
print(x)

##################

a = (1, 5, 3, 9)
x = max(a)
print(x)
```

```
10
Vicky
9
```

```
def maximum(a, n):
    maxpos = a.index(max(a))
    print ("The maximum is at position", maxpos + 1)

a = [3, 4, 1, 3, 4, 5]
maximum(a, len(a))
```

```
The maximum is at position 6
```



## Python min() Function

- The min() function returns the item with the lowest value, or the item with the lowest value in an iterable.
- If the values are strings, an alphabetically comparison is done.
- Syntax

min(*n1, n2, n3, ...*)

Or

min(*iterable*)

## Python min() Function - Examples

```
x = min(5, 10)
print(x)

##################

x = min("Mike", "John", "Vicky")
print(x)

##################

a = (1, 5, 3, 9)
x = min(a)
print(x)

##################

a = ["Alice", "Bob", "Christina", "Dan"]

res = min(a, key=len)
print(res)
```

```
5
John
1
Bob
```



## Python print() Function

- The print() function prints the specified message to the screen, or other standard output device.
- The message can be a string, or any other object, the object will be converted into a string before written to the screen.
- Syntax

  print*(object(s)*, sep=*separator*, end=*end*, file=*file*, flush=*flush*)

Where

sep='separator' is optional that specify how to separate the objects, if there is more than one. The default is ' '

end='end' is optional that specify what to print at the end. The default is '\n' (line feed)

## Python print() Function - Examples

```
print("Hello World")

######################

print("Hello", "how are you?")

######################

print("Hello", "how are you?", sep="---")

######################

print("OK", end=" ")
print("See you")
```

```
Hello World
Hello how are you?
Hello---how are you?
OK See you
```



## Python range() Function

- The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number.
- Syntax

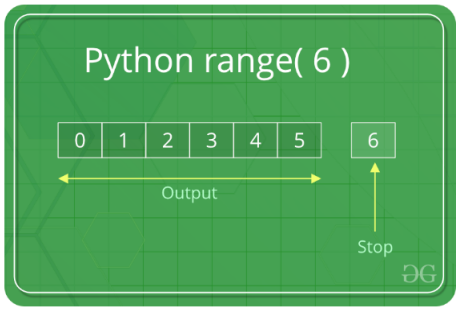
range(*start, stop, step*)

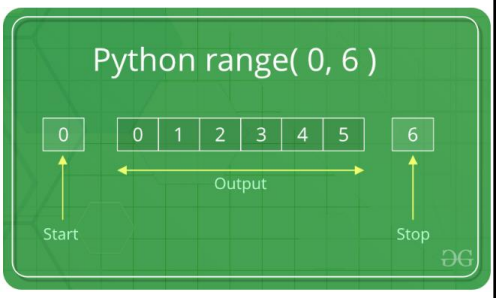
Where

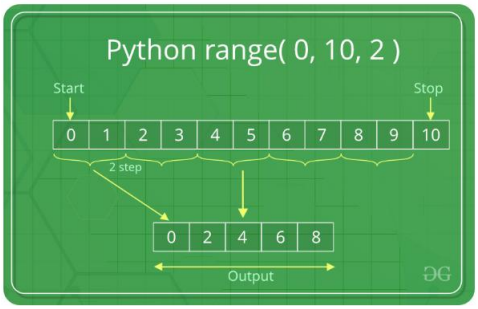
Start is optional. An integer number specifying at which position to start. Default is 0

Stop is required. An integer number specifying at which position to stop (not included).

Step is optional. An integer number specifying the incrimination. Default is 1






## Python range() Function

```
x = range(6)
for n in x:
  print(n, end=" ")
print("\n")

########################

x = range(3, 6)
for n in x:
  print(n, end=" ")
print("\n")

########################

x = range(3, 20, 5)
for n in x:
  print(n, end=" ")
```

```
0 1 2 3 4 5

3 4 5

3 8 13 18
```

## Python reversed() Function

- The reversed() function returns a reversed iterator object.
- Syntax

  reversed(*sequence*)

Where sequence is required. Any iterable object.

```
alph = ["a", "b", "c", "d"]

ralph = reversed(alph)

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

```
d
c
b
a
```



## Python sorted() Function

- The sorted() function returns a sorted list of the specified iterable object.
- You can specify ascending or descending order. Strings are sorted alphabetically, and numbers are sorted numerically.
- You cannot sort a list that contains BOTH string values AND numeric values.
- Syntax

  sorted(*iterable*, key=*key*, reverse=*reverse*)

Where

Iterable is required. The sequence to sort, list, dictionary, tuple etc.

Key is optional. A Function to execute to decide the order (Default is None)

Reverse is optional. A Boolean. False will sort ascending, True will sort descending (Default is False)

```
a = ("b", "g", "a", "d", "c", "h")
x = sorted(a)
print(x)

############################

a = (1, 11, 2)
x = sorted(a)
print(x)

############################

a = ("h", "b", "c", "a", "d", "g")
x = sorted(a, reverse=True)
print(x)

############################

a = ("Jenifer", "Sally", "Jane")
x = sorted(a, key=len)
print(x)

############################

def myfunc(n):
  return abs(10-n)

a = (5, 3, 1, 11, 2, 12, 17)
x = sorted(a, key=myfunc)
print(x)
```

```
['a', 'b', 'c', 'd', 'g', 'h']
[1, 2, 11]
['h', 'g', 'd', 'c', 'b', 'a']
['Jane', 'Sally', 'Jenifer']
[11, 12, 5, 3, 17, 2, 1]
```

Python range() Function

25

# Python sum() Function

- The sum() function returns a number, the sum of all items in an iterable.
- Syntax

sum(*iterable, start*)

Where

Iterable is required. The sequence to sum

Start is optional. A value that is added to the return value

```
a = (1, 2, 3, 4, 5)
x = sum(a)
print(x)

#####################

a = (1, 2, 3, 4, 5)
x = sum(a, 7)
print(x)
```

```
15
22
```

26

by: Fatima Ben Lashihar

## Python zip() Function

- The zip() function returns a zip object, which is an iterator of tuples where the first item in each passed iterator is paired together, and then the second item in each passed iterator are paired together etc.
- If the passed iterables have different lengths, the iterable with the least items decides the length of the new iterator.
- Syntax

  zip(*iterator1, iterator2, iterator3 ...*)



## Python zip() Function

```
a = ("John", "Charles", "Mike")
b = ("Jenny", "Christy", "Monica")

x = zip(a, b)

print(tuple(x))
```

```
(('John', 'Jenny'), ('Charles', 'Christy'), ('Mike', 'Monica'))
```

# The END