### 19CSE201: Advanced Programming

# Lecture 20 Loops, Functions & Exception Handling in Python

By
Ritwik M
Assistant Professor(SrGr)
Dept. Of Computer Science & Engg

## A Quick Recap

- · Anatomy of Python
- · Python Data Types
- · Operators in python
- · Conditional Statements

### Loops

- · Python has 2 primitive loop commands
  - · While
  - for

### The while Loop

• With the while loop we can execute a set of statements as long as a condition is true.

```
• i = 1
  while i < 6:
    print(i)
    i += 1</pre>
```

• Can be combined with the else statement and we can run a block of code once when the condition no longer is true:

```
• i = 1
  while i < 6:
    print(i)
    i += 1
else:
    print("i is no longer less than 6")</pre>
```

### The while Loop - Other operations

- · Continue
- · Example

```
• i = 0
while i < 6:
    i += 1
    if i == 3:
        continue
    print(i)</pre>
```

- · Break
- · Example

```
• i = 0
while i < 6:
    i += 1
    if i == 3:
        break
    i+=1
    print(i)</pre>
```

### The for Loop

- A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).
- This is less like the for keyword in other programming languages and works more like an iterator method as found in other object-orientated programming languages.
- With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.
- The for loop does not require an indexing variable to set beforehand.
- · Example: Loop through lists
  - fruits = ["apple", "banana", "cherry"]
     for x in fruits:
     print(x)

### The for Loop - Other operations

- · Looping Through a String
  - for x in "banana":
     print(x)
- · Break and Continue

```
• fruits = ["apple",
  "banana", "cherry"]
  for x in fruits:
    if x == "banana":
        break
    print(x)
```

- · Else in For Loop
  - The else keyword in a for loop specifies a block of code to be executed when the loop is finished:
    - for x in range(6):
       print(x)
      else:
       print("Finished!")

### The for Loop - Other operations

- The range () Function
- To loop through a set of code a specified number of times, we can use the range() function,
- The range () function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.
- for x in range (6):
  - print(x)
- Using the start parameter:
  - for x in range(2, 6): print(x)

- The pass statement
  - for loops cannot be empty, but if you for some reason have a for loop with no content, put in the pass statement to avoid getting an error.
    - for x in [0, 1, 2]: pass

What about nested loops? Try it..!

#### Functions

- · A function is a block of code which only runs when it is called.
- · You can pass data, known as parameters, into a function.
- · A function can return data as a result.
- · Parameters or Arguments?
  - The terms parameter and argument can be used for the same thing: information that are passed into a function.
  - From a function's perspective:
    - A parameter is the variable listed inside the parentheses in the function definition.
    - An argument is the value that is sent to the function when it is called.

Creating a Function
 In Python a function is defined using the def keyword:
 def my function():
 print("Hello from a function")

- · Calling a Function
  - · To call a function, use the function name followed by parenthesis:

```
• def my function():
    print("Hello from a function")
    my_function()
```

- Arguments
  - Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

```
• def my function(firstname):
    print(fname + " someLastName")
    my function("Name1")
    my_function("Name2")
    my_function("Name3")
```

- · Arbitrary Arguments, \* args
  - If you do not know how many arguments that will be passed into your function, add a \* before the parameter name in the function definition.
  - This way the function will receive a tuple of arguments, and can access the items accordingly

```
• def my function(*kids):
    print("The youngest child is " + kids[2])

my_function("kid1", "kid2", "kid3")
```

- · Keyword Arguments
  - You can also send arguments with the key = value syntax. This ignores the arg order
    - def my\_function(child3, child2, child1):
       print("The youngest child is " + child3)

      my\_function(child1 = "kid1", child2 = "kid2", child3 = "kid3")

- · Passing a List as an Argument
  - You can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.

```
• def my_function(food):
    for x in food:
        print(x)

fruits = ["apple", "banana", "cherry"]

my_function(fruits)
```

- Return Values
  - To let a function return a value, use the return statement
    - def my\_function(x):
       return 5 \* x

      print(my\_function(3))
      print(my\_function(5))

- · Recursion
  - In this example, tri\_recursion() is a function that we have defined to call itself ("recurse"). The k variable is used as the data, which decrements (-1) every time we recurse.
  - The recursion ends when the condition is not greater than o (i.e. when it is o).

```
• def tri_recursion(k):
    if(k > 0):
        result = k + tri_recursion(k - 1)
        print(result)
    else:
        result = 0
    return result

print("\n\nRecursion Example Results")
    tri recursion(6)
```

### Exception Handling

- When an error occurs, or exception as we call it, Python will normally stop and generate an error message.
- · The try block lets you test a block of code for errors.
- · The except block lets you handle the error.
- The finally block lets you execute code, regardless of the result of the try- and except blocks.

## Handling Exceptions

- The try block will generate an exception, because x is not defined:
  - try:
     print(x) # will raise an error as x is not defined
     except:
     print("An exception occurred")
- Multiple Exceptions
  - Print one message if the try block raises a Name Error and another for other errors:

```
• try:
    print(x)
except NameError:
    print("Variable x is not defined")
except:
    print("Something else went wrong")
```

## Handling Exceptions Cont

• You can use the else keyword to define a block of code to be executed if no errors were raised:

```
• try:
    print("Hello")
    except:
    print("Something went wrong")
    else:
        print("Nothing went wrong")
```

## Handling Exceptions Cont

• The finally block, if specified, will be executed regardless if the try block raises an error or not.

```
• try:
    print(x)
except:
    print("Something went wrong")
finally:
    print("The 'try except' is finished")
```

• This can be useful to close objects and clean up resources:

### Raising an exception

- As a Python developer you can choose to throw an exception if a condition occurs.
- To throw (or raise) an exception, use the raise keyword.
  - Example: Raíse an error and stop the program if x is lower than 0:

```
• x = -1
if x < 0:
   raise Exception("Sorry, no numbers below zero")</pre>
```

• You can define what kind of error to raise, and the text to print to the user.

```
• x = "hello"
if not type(x) is int:
   raise TypeError("Only integers are allowed")
```

### Exercise

- · Spot and fix the error in this code
  - def my\_function(fname, lname):
     print(fname + " " + lname)

```
my_function("Emil")
```

• Hint - Look at the arguments

# Quíck Summary

- · Loops
- · For and While
- Functions
- Arguments
- Recursion
- Examples
- Exercises

# up Next

Namespaces & Scopes