Atmiya University
Faculty of Science,
Department of Computer Science & I.T.

Subject Name: 21UFSDE309 Data Science Using Python

Python Conditions

- **If statements**
- **Python if Statement** is used for decision-making operations. It contains a body of code which runs only when the condition given in the if statement is true.
- **Python if Statement Syntax:**
  - if expression :
    - Statement

Python Conditions

- **Example**
- a = 33

```
b = 200
if b > a:
  print("b is greater than a")
```

Python Conditions

- **Indentation**
- Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly-brackets for this purpose.
- **Example**
- If statement, without indentation (will raise an error):

-
  ```
  a = 33
  b = 200
  if b > a:
  print("b is greater than a") # you will get an error
  ```

Python Conditions

- **Else**
- The else keyword catches anything which isn't caught by the preceding conditions.
-
```
a = 200
b = 33
if b > a:
  print("b is greater than a")
else:
  print("a is greater than b")
```

Python Conditions

- **Elif**
- The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".
-
```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

Python Conditions

- **Elif**
- The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".
-
```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

Python Loops

- Python has two primitive loop commands:
  - while loops
  - for loops
- **The while Loop**
- With the while loop we can execute a set of statements as long as a condition is true.
- **Example**
- Print i as long as i is less than 6:

```
i = 1
while i < 6:
  print(i)
  i += 1
```

The break Statement

- With the break statement we can stop the loop even if the while condition is true:
- **Example**
- Exit the loop when i is 3:
-
```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

The continue Statement

- With the continue statement we can stop the current iteration, and continue with the next:
- **Example**
- Continue to the next iteration if i is 3:
-
```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```
- # Note that number 3 is missing in the result

The else Statement

- With the else statement we can run a block of code once when the condition no longer is true:
- **Example**
- Print a message once the condition is false:

-
  ```
  i = 1
  while i < 6:
    print(i)
    i += 1
  else:
    print("i is no longer less than 6")
  ```

Python For Loops

A for loop is used for iterating over a sequence (that is a list, a tuple, a dictionary, a set, or a string).

- **Example**
- Print each fruit in a fruit list:
-
```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

Python For Loops

- **Looping Through a String**
- Even strings are iterable objects, they contain a sequence of characters:
- **Example**
- Loop through the letters in the word "banana":

-
  ```
  for x in "banana":
    print(x)
  ```

Python For Loops

- **The break Statement**
- With the break statement we can stop the loop before it has looped through all the items:
- **Example**
- Exit the loop when x is "banana":

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

Python For Loops

- **The continue Statement**
- With the continue statement we can stop the current iteration of the loop, and continue with the next:
- **Example**
- Do not print banana:
-
```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    continue
  print(x)
```

Python For Loops

- **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.
- **Example**
- Using the range() function:
-
```
for x in range(6):
 print(x)
```

Python For Loops

- The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(2, 6), which means values from 2 to 6 (but not including 6):
- **Example**
- Using the start parameter:
- for x in range(2, 6):
  print(x)

Python For Loops

- The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, **3**):
- **Example**
- Increment the sequence with 3 (default is 1):
- for x in range(2, 30, 3):
  print(x)

Python For Loops

- **Else in For Loop**
- The else keyword in a for loop specifies a block of code to be executed when the loop is finished:
- **Example**
- Print all numbers from 0 to 5, and print a message when the loop has ended:

-
  ```
  for x in range(6):
    print(x)
  else:
    print("Finally finished!")
  ```

Python For Loops

- **Nested Loops**
- A nested loop is a loop inside a loop.
- The "inner loop" will be executed one time for each iteration of the "outer loop":
- **Example**
- Print each adjective for every fruit:
-
```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)
```

Python assert Keyword

- **Definition and Usage**
- The assert keyword is used when debugging code.
- The assert keyword lets you test if a condition in your code returns True, if not, the program will raise an AssertionError.
- You can write a message to be written if the code returns False, check the example below.

Python assert Keyword

- **Example**
- Test if a condition returns True:

```
x = "hello"
#if condition returns True, then nothing happens:
assert x == "hello"
#if condition returns False, Assertion Error is raised:
assert x == "goodbye"
```

Python assert Keyword

- **Example**
- Write a message if the condition is False:
- x = "hello"

```
#if condition returns False, AssertionError is raised:
assert x == "goodbye", "x should be 'hello'"
```

Python pass Statement

- **Definition and Usage**
- The pass statement is used as a placeholder for future code.
- When the pass statement is executed, nothing happens, but you avoid getting an error when empty code is not allowed.
- Empty code is not allowed in loops, function definitions, class definitions, or in if statements.

Python pass Statement

- **Example**
- Create a placeholder for future code:
- for x in [0, 1, 2]:
   pass

Python Arrays

- Arrays are used to store multiple values in one single variable:
- **Example**
- Create an array containing car names:

```
cars = ["Ford", "Volvo", "BMW"]
```

Python Arrays

- **What is an Array?**
- An array is a special variable, which can hold more than one value at a time.
- If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:
- car1 = "Ford"
  car2 = "Volvo"
  car3 = "BMW"
- However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?
- The solution is an array!
- An array can hold many values under a single name, and you can access the values by referring to an index number.

Python Arrays

- **Access the Elements of an Array**
- You refer to an array element by referring to the *index number*.
- **Example**
- Get the value of the first array item:

```
x = cars[0]
```

Python Arrays

- **Example**
- Modify the value of the first array item:
  - cars[0] = "Toyota"
- **The Length of an Array**
- Use the len() method to return the length of an array (the number of elements in an array).
- **Example**
- Return the number of elements in the cars array:
- x = len(cars)

Python Arrays

- **Looping Array Elements**
- You can use the for in loop to loop through all the elements of an array.
- **Example**
- Print each item in the cars array:
- for x in cars:

  print(x)

Python Arrays

- **Adding Array Elements**
- You can use the append() method to add an element to an array.
- **Example**
- Add one more element to the cars array:
- cars.append("Honda")

Python Arrays

- **Removing Array Elements**
- You can use the pop() method to remove an element from the array.
- **Example**
- Delete the second element of the cars array:
- cars.pop(1)

Python Arrays

- You can also use the remove() method to remove an element from the array.
- **Example**
- Delete the element that has the value "Volvo":
- cars.remove("Volvo")

Python Arrays

- **Array Methods**
- Python has a set of built-in methods that you can use on lists/arrays.
- **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 first item with the specified value
- **reverse()** Reverses the order of the list
- **sort()** Sorts the list

Functions

- A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.
- 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.
- Python gives you many built-in functions like print(), etc. but you can also create your own functions. These functions are called user-defined functions.

Defining a Function

- You can define functions to provide the required functionality. Here are simple rules to define a function in Python.
- Function blocks begin with the keyword def followed by the function name and parentheses (( )).
- Any input parameters or arguments should be placed within these parentheses.
- ??You can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or docstring.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

Functions

Syntax:

```
def functionname( parameters ):
 "function_docstring" - Optional
 function_suite
 return [expression]
```

By default, parameters have a positional behavior and you need to inform them in the same order that they were defined.

Creating Functions

In Python a function is defined using the def keyword:

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

Calling Functions

To call a function, use the function name followed by parenthesis:

```
def fun():
    print("Hello");
fun()
```

Return result from a function

A return statement is used to end the execution of the function call and "returns" the result (value of the expression following the return keyword) to the caller. The statements after the return statements are not executed.

If the return statement is without any expression, then the special value None is returned. A return statement is overall used to invoke a function so that the passed statements can be executed.

Syntax:

```
def fun():
    statements
    .
    return [expression]
```

Example:

```
def add(a, b):
    return a + b

def is_true(a):
    return bool(a)

res = add(2, 3)
print("Result of add function is {}".format(res))
res = is_true(2<5)
print("Result of is_true function is {}".format(res))
```

Return multiple values from a function:

In Python, we can return multiple values from a function. Following are different ways.

Using Object: This is similar to C/C++ and Java, we can create a class (in C, struct) to hold multiple values and return an object of the class.

Example:

```
class Test:
    def __init__(self):
        self.str = "Atmiya University"
        self.x = 20
def fun():
    return Test()
t = fun()
print(t.str)
print(t.x)
```