A Refresher on Python

Python’s conditional statements

The conditional statements in Python are: if, elif, and else

What's the output of this code snippet?

```
x = 0
if x == 3:
    print('x is equal to 3')
else:
    print('x is NOT equal to 3')
print('That\'s it!')
```

Python does not have switch statements!

Python’s Loops

While Loop

```
n=10
while n > 0:
    print(n)
    n = n-1
print('Complete!')
```

For Loop

Run these code and see the output

```
languages = ['Arabic', 'English', 'French', 'Spanish']
for lang in languages:
    print ('This language is in the list: ' +  lang)
```

```
languages = ['Arabic', 'English', 'French', 'Spanish']
counter = 0
for lang in languages:
    print ('This language is in the list: ' +  languages[counter])
    counter = counter + 1
```

What will be the output of the following code

```
for x in range(1,5):
    print(x)
```

Statements Used in While and For Loops

break

The statement break causes the loop to terminate, and program execution is continued on the next statement.

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
value = 1
while value in numbers:
    if value == 5:
        break
    print('I\'m # ' + str(value))
    value = value + 1

print('Sorry, I had to quit the loop when the value became 5')
```

<u>**Output**</u>:

```
I'm # 1
I'm # 2
I'm # 3
I'm # 4
Sorry, I had to quit the loop when the value became 5
```

continue

This statement returns control back to the beginning of the loop, ignoring any statements in the loop coming afterward.

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
value = 1
while value in numbers:
    if value < 5:
        print('I\'m # ' + str(value))
        value = value + 1
        continue
        print('I\'m in the if-condition, why are you ignoring me?!')
    elif value == 5:
        break
```

```
print('I have reached the last statement in the program and need to exit')
```

Output:

```
I'm # 1
I'm # 2
I'm # 3
I'm # 4
I have reached the last statement in the program and need to exit
```

pass

The pass statement is a null statement, that is it doesn't do anything. Suppose you were writing a program, and at some point you weren't sure what should go in the for-statement for instance, as follows:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for value in numbers:
    pass  # not sure what to do here yet

print('I have reached the last statement in the program and need to exit')
```

Output:

```
I have reached the last statement in the program and need to exit
```

else

The else statement contain a block of statements to run when the loop exits in a normal way, and not by a break.

```
numbers = [1,2,3]
value = 1
while value in numbers:
        print ('I\'m # ' + str(value))
        value = value + 1
else:
    print('I\'m part of the else statement block')
    print('I\'m also part of the else statement block')
```

Output:

```
I'm # 1
I'm # 2
I'm # 3
I'm part of the else statement block
I'm also part of the else statement block
```

Python’s Functions

Functions are composed of a set of instructions combined together to get some result (achieve some task), and are executed by calling them, namely by a function call.
Functions can be either built-in functions (mentioned above) or user-defined functions.
In Python, you may have come across things like file(), print(), open(), range(), etc. Those are called built-in functions. That is, functions already provided by the language itself which you can execute by referencing (calling) to them.

Defining Functions

syntax:
```
def function_name(parameters):
    statement(s)
    return expression
```

Example, if we would like to display the name of any employee entered in the system;

```
employee_name = 'xyz'
def print_name(name):
    print(name)
print_name(employee_name)
```

Try this program and see the output:

```
def add(x, y):
    return x + y


def subtract(x, y):
    return x - y


def multiply(x, y):
    return x * y


def divide(x, y):
    return x / y


x = 8
y = 4

print('%d + %d = %d' % (x, y, add(x, y)))
print('%d - %d = %d' % (x, y, subtract(x, y)))
print('%d * %d = %d' % (x, y, multiply(x, y)))
print('%d / %d = %d' % (x, y, divide(x, y)))
```

Standard datatypes in Python

1. Numbers
2. String
3. List
4. Tuple
5. Dictionary

1. <u>Numbers</u>

Number stores numeric values. Python creates Number objects when a number is assigned to a variable. For example;

a = 3 , b = 5 #a and b are number objects

Python supports 4 types of numeric data.

1. int
2. long
3. float
4. complex (complex numbers like 2.14j, 2.0 + 2.3j, etc.)

2. <u>String</u>

The string can be defined as the sequence of characters represented in the quotation marks. In python, we can use single or double or triple quotes to define a string.

In the case of string handling, the operator + is used to concatenate two strings as the operation "hello"+" python" returns "hello python".

The operator * is known as repetition operator as the operation "Python " *2 returns "Python Python ".

<u>Example:</u>

```
str1='hello'#string str1
str2=' how are you'#string str2
print(str1[0:2])#printing first two character using slice operator
print(str1[4])#printing 4th character of the string
print(str1*2)#printing the string twice
print(str1+str2)#printing the concatenation of str1 and str2
```

<u>Output</u>:

```
he
o
hellohello
hello how are you
```

3. List

Lists are similar to arrays in C. However; the list can contain data of different types. The items stored in the list are separated with a comma (,) and enclosed within square brackets [].

Eg: b = [1, "hi", "python", 2]
Accessing Items
Lists make it easy to access items regardless of the number of items you have in the list. This is done using an index.
For instance, our list ‘b’ consists of 4 items. In Python, indexing starts from the value 0. So, the first item in the list will have index 0, the second item index 1, and so forth.
Let's say we want to access the last item in this list. This can be simply done as follows:
b[3]

List Operations

(i) We can use **slice** [:] operators to access the data in the list.
(ii) The **concatenation** operator (+) and **repetition** operator (*) works with the list in the same way as they were working with the strings.
(iii) To remove an element from the list, **del** function can be used
(iv) A new element can be added to the list using **append**() function

Example:

```
b=[1,"hi","python",2]
print(b[3:])#prints the list from index 3 to end of the list
print(b[0:2])#prints from the beginning of the list to element at index 2
print(b)
del b[1]#deletes the element at index 1
print(b)
b.append("new")#appends a string "new" to the end of the list
print(b)
print(b+b)#concatenates two lists
print(b*3)#repeats the list 3 times
```

Output:

```
[2]
[1, 'hi']
[1, 'hi', 'python', 2]
[1, 'python', 2]
[1, 'python', 2, 'new']
[1, 'python', 2, 'new', 1, 'python', 2, 'new']
[1, 'python', 2, 'new', 1, 'python', 2, 'new', 1, 'python', 2, 'new']
```

4. Tuple

A tuple is similar to the list in many ways. Like lists, tuples also contain the collection of the items of different data types.
There are two main differences between list and tuple:

1. Tuples are immutable, so once you create a Tuple, you cannot change its content or even its size, unless you make a copy of that Tuple.
2. They are written in parentheses ( ) rather than in square brackets [ ]

Example:

```
t=("hi", "python", 2)
print(t[1:])
print(t[0:1])
print(t)
print(t+t)
print(t*3)
print(type(t))
```

Output:

```
('python', 2)
('hi',)
('hi', 'python', 2)
('hi', 'python', 2, 'hi', 'python', 2)
('hi', 'python', 2, 'hi', 'python', 2, 'hi', 'python', 2)
<class 'tuple'>
```

Try this code and understand the output:

```
t=("hi", "python", 2)
t[2]="hello"
```

5. Dictionary

You can think of a dictionary as a mapping between a set of indices (which are called keys) and a set of values. Each key maps to a value. The association of a key and a value is called a key-value pair or sometimes an **item**.
The items in the dictionary are separated with the comma and enclosed in the curly braces {}.

The function dict creates a new dictionary with no items. Because dict is the name of a built-in function, you should avoid using it as a variable name.

```
>>> print(eng2sp)
{}
```

The squiggly-brackets, {}, represent an empty dictionary. To add items to the dictionary, you can use square brackets:

```
>>> eng2sp['one'] = 'uno'
```

This line creates an item that maps from the key 'one' to the value 'uno'.

The order of the key-value pairs might not be the same in dictionary. In general, the order of items in a dictionary is unpredictable.

But that's not a problem because the elements of a dictionary are never indexed with integer indices. Instead, you use the keys to look up the corresponding values:

eg:

```
print(eng2sp['one'])
'uno'
```

The len function works on dictionaries; it returns the number of key-value pairs:

```
>>> len(eng2sp)
3
```

The in operator works on dictionaries; it tells you whether something appears as a key in the dictionary

```
>>> 'one' in eng2sp
True
>>> 'uno' in eng2sp
False
```

To see whether something appears as a value in a dictionary, you can use the method values, which returns the values as a list, and then use the in operator:

```
>>> vals = eng2sp.values()
>>> 'uno' in vals
True
```

Try this program and understand the output

```
engsp = dict()
print(engsp)
engsp['one'] = 'uno'
engsp['two'] = 'dos'
engsp['three'] = 'tres'
print(engsp)
print(len(engsp))
print('one' in engsp)
```

Example 1: Python program to count the number of characters (character frequency) in a string.

```
def char_frequency(str1):
    dict = {}
    for n in str1:
        keys = dict.keys()
        if n in keys:
            dict[n] += 1
        else:
            dict[n] = 1
    return dict
print(char_frequency('google.com'))
```

Example 2: Python program to copy the contents of a File to another file

```
fp=open("file2.txt","w")#open a file2 in write mode
fp1=open("file1","r")#open a file1 in read mode
reading_file=fp1.read() #Read contents of a first file, store it in a string
fp.write(reading_file)#Write the string to second file
```

Example 3: Python program to count the number of words in a text file .

```
fname = input("Enter the file name:")
wc = 0
fp = open(fname, "w")
fp.write("My first line\n")
fp.write("This is the next line")
fp.close()
fp = open(fname, "r")
for line in fp:
    #print(line)
    word = line.split()
    wc=wc+len(word)
    #print(word)
print(wc)
```