Pyhton Basics

Author	Shara
Property	Notes

Basic Syntax

Let’s say the vaiable is

```
string = "Giraffe Academy"
print(phrase.replace("Giraffe", "Elephant"))
# replaces the word Giraffe with Elephant
```

- *string*.upper() - converts all the characters in the string in upper cases
- *string*.isupper() - tells if the characters in the given string are in upper or lower cases - returns value by - either True or False
- len(*string*) - tells the length of the string
- *string*.index(”character name, like a, b, acad etc”) - returns the index of the position of the character in the string
- *string*.replace() - replaces the selected character or sting with another

```
print(3*4 + 5)
print(10 % 3) # Modulus operator - gives remainder
my_num = 5
print(my_num)
print(str(my_num)) # converts the number into a string
print(abs(my_num)) # gives absolute number
print(pow(3, 2)) # 3 raised to the power 2, 3^2
print(max(num1, num2) # gives max of the two
# round(), min()
```

- Inputting values from user

  ```
  name = input("Enter your name: ")
  age = input("Enter your age: ")
  ```

  For example,

  ```
  num1 = input("")
  num2 = input("")
  result = num1 + num2
  # The result basically comes as num1num2, i.e., for 5 and 3,
  # the result would come as 53 and not as 8
  ```

- So, to get rid of this problem, we us functions like *int(), float()*, that will convert the variables around to int or float. **Float** takes decimal numbers, whereas, **int** takes whole numbers.

```
result = float(num1) + float(num2)
```

Lists

```
friends = ["Kevin", "Karen", "Jim", "Oscar", 'Toby']
#             0        1       2       3        4
#            -5       -4      -3      -2       -1
print(friends)
print(friends[2])
print(friends[-1])
print(friends[1:3]) # grabs elements indexing at 1 and 2, and not including 3
# Output: ['Karen', 'Jim']
```

Functions involving Lists

```
lucky_numbers = [4, 8, 15, 16, 23, 42]
friends = ["Kevin", "Karen", "Jim", "Oscar", 'Toby']
friends.extend(lucky_numbers)
print(friends)
```

Other functions:

```
friends.append("Creed") # This function always adds in the end of the list
friends.insert(1, "Kelly") # will put Kelly at index = 1, and push
# others one index ahead
friends.remove("Jim")
friends.clear() # Output: []
friends.pop() # removes the last element from the list
print(friends.index("Karen") # returnss the index from the list
print(friends.count("Jim")) # counts the number of times the element occurs
```

Sorting Function

```
lucky_numbers.sort() # sorts the list in ascending order
lucky_numbers.reverse() # Reverses the order of list
print(lucky_numbers)

friends2 = friends.copy() # Copies the friends list
```

Tuples (a Data Structure)

Tuples are similar to lists - storing information. They are immutable, i.e., cannot be changed or modified. That's the only basic difference between tuples and lists. And that's why, coordinates, would be best used as tuples.

```
coordinates = (4, 5)
print(coordinates[0])
coordinated[1] = 10 # Output: shows error

positions = [(4, 5), (6, 7), (80, 34)] # tuples inside lists
```

Functions

Help in organising codes.

```
def say_hi(name, age):
  print("Hello" + name + ", you are " + age)

say_hi("Mike", "35")
```

```
def cube(num):
  return num*num*num
```

If/else statements

```
is_male = True
is_tall = False
if is_male and is_tall:
  print("You are a tall male")
elif is_male and not(is_tall):
  print("You are a short male")
else:
  print("You are neither a male nor tall")
```

```
def max_num(num1, num2, num3):
  if num1 >= num2 and num1>= num3:
    return num1
  elif num2 >= num1 and num2 >= num3:
    return num2
  else:
    return num3

print(max_num(300, 40, 5))
```

Dictionaries

As in a normal dictionary, where there is a word and its definition is present. So here, the '*word*' is a '*key*', and the '*value;* would be actually its '*definition'*. We can access values from the dictionary created.

```
daysConversions = {
  "Mon" : "Monday",
  "Tue" : "Tuesday",
  "Wed" : "Wednesday",
  "Thur" : "Thursday",
  "Fri" : "Friday",
  "Sat" : "Saturday",
  "Sun" : "Sunday"
}

print(daysConversions["Thur"])
print(daysConversions.get("Mon", "Not a valid key"))
print(daysConversions.get("Luv", "Not a valid key"))
# Instead of 'none', if the key is not present in the dictionary, it will print out
# "Not a valid key"
```

While Loops

```
i = 1
while i <= 10:
  print(i)
  i = i + 1
print("Done with loop")
```

For Loops

```
friends = ["Jim", "Karen", "Kevin"]
len(friends)
for letter in "Giraffe Academy":
  print(letter)
for friend in friends:
  print(friend)
for index in range(len(friends)):
  print(friends[index])
for index in range(3, 10):
  print(index) # prints all nos. from 3 to 9
```

Exponent Functions

```
num = 2**3
def raise_to_power(base_num, pow_num):
  result = 1
  for index in range(pow_num):
    result = result*base_num
  return result
print(raise_to_power(2, 3)) # 2^3
```

2D Lists/Nested Loops

```
number_grid = [
                [1, 2, 3],
                [4, 5, 6],
                [7, 8, 9],
                [0]
              ]
# 4 rows, 3 columns
print(number_grid[0][0]) # Output: 1
print(number_grid[2][1]) # Output: 8
for row in number_grid:
  for col in row:
    print(col)
```

Importing Functions from Modules

- There are several functions in python, mathematical operations that can be used, for which, we need to import external code through modules

```
from math import *
# helps in grabbing different mathematical functions

print(floor(3.7)) # this basically chops off the no. after decimal and returns the lowest no.
print(ceil(3.2)) # rounds up and returns the max value, answer = 4
print(sqrt(36)) # returns the square root, i.e., 6
```

Try/Except Block

Helps in debugging. If the code is correct, helps in smooth running, otherwise, shows the error explicitly.

```
try:
  number = int(input("Enter a number: ))
  print(number)
except:
  print("Invalid Input")
# it breaks the program, and shows this line instead
```

For example,

```
try:
  value = 10/0
  number = int(input("Enter a number: ))
  print(number)
except:
  print("Invalid Input")
```

Solution for the same:

```
try:
  value = 10/0
  number = int(input("Enter a number: ))
  print(number)
except ZeroDivisionError:
  print("Divided by zero")
except ValueError:
  print("Invalid Input")
```

Reading Files

```
employee_file = open("employees.txt", "r")
# r: read, w: write, a: append: add new info in the end of the file

print(employee_file.readable()) # this function helps to know if the files that we are trying to open are readable or not
# returns a boolean value

print(employee_file.read())

print(employee_file.readline()) # returns each line, line by line

# print(employee_file.readlines()[1]) # reads all the lines and returns them as arrays

# Or we can do the following (works same as readlines()): -
for employee in employee_file.readlines():
    print(employee)

employee_file.close()
```

Writing Files

```
employee_file = open("employees.txt", "a")

# employee_file = open("employees.txt", "w")
# write overwrites everything into whatever we'll write right now
```

```
employee_file.write("\nToby - Human Resources")
# employee_file.write("\nKelly - Customer Service")


employee_file.close()

new_file = open("index.html", "w")

new_file.write("<p>This is HTML!</p>")
```

Classes and Objects

```
class Student:
# initialize function
    def __init__(self, name, major, gpa, is_on_probation):
        self.name = name
        self.major = major
        self.gpa = gpa
        self.is_on_probation = is_on_probation
```

```
from Student import Student

student1 = Student("Jim", "Business", 3.1, False)
student2 = Student("Dwight", "Business", 3.7, True)
# Object is just an instance of a class
# student1 and student2 are objects here

print(student1.name)
```

A class function is just a little function that can be used by the objects of the class.

Inheritance

This is where we can define a bunch of attributes and functions and things inside the class and then we can create another class and we can inherit all of those attributes.

```
class Chef:

    def make_chicken(self):
        print("The chef makes a chicken")

    def make_salad(self):
        print("The chef makes a salad")

    def make_special_dish(self):
        print("The chef makes bbq ribs")
```

```
from chef import Chef

class ChineseChef(Chef):
    # def make_chicken(self):
    #     print("The chef makes a chicken")
    # def make_salad(self):
    #     print("The chef makes a salad")

    # def make_special_dish(self):
    #     print("The chef makes orange chicken")

    # So, instead of writing all of the above code, we can do one small thing,
    # that is inherit
    # We can write the top two lines of codes instead
    def make_special_dish(self):
        print("The chef makes orange chicken")
# since, the chef file also had make_special_dish function, so, if
# we would have run the file, make_special_dish function would have showed the result
# of the Chef class, so, we'll have to overwrite the same function to get the
# special dish of ChineseChef


    def make_fried_rice(self):
        print("The chef makes fried rice")
```

```
from chef import Chef
from ChineseChef import ChineseChef

myChef = Chef()
myChef.make_special_dish()

myChineseChef = ChineseChef()
myChineseChef.make_special_dish()
```

Python Interpreter

It’s a little environment that we can use to execute python commands. So, it’s kind of this little sandbox environment where we can test out and try out different python commands and different python functions, in a very safe and neutral environment.