Programming with Python 3

Introduction to Variables

- Harsh Nandwana

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.

Example:

```
print("Hello World")
```

Syntax

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

Parameter Values

Parameter	Description
object(s)	Any object, and as many as you like. Will be converted to string before printed
sep='*separator*'	Optional. Specify how to separate the objects, if there is more than one. Default is ' '
end='*end*'	Optional. Specify what to print at the end. Default is '\n' (line feed)
file	Optional. An object with a write method. Default is sys.stdout
flush	Optional. A Boolean, specifying if the output is flushed (True) or buffered (False). Default is False

Variables

Variables are containers for storing data values.

Creating Variables

- Python has no command for declaring a variable.
- A variable is created the moment you first assign a value to it.

```
x = 5
y = "John"
print(x)
print(y)
```

- Variables do not need to be declared with any particular *type*, and can even change type after they have been set.

```
x = 4       # x is of type int
x = "Sally" # x is now of type str
print(x)
```

Python Variables - Assign Multiple Values

- **Many Values to Multiple Variables**
  - Python allows you to assign values to multiple variables in one line:

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

- **One Value to Multiple Variables**
  - And you can assign the *same* value to multiple variables in one line:

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

- **Unpack a Collection**
  - If you have a collection of values in a list, tuple etc. Python allows you extract the values into variables. This is called *unpacking*.

```
fruits = ["apple", "banana", "cherry"]
x, y, z = fruits
print(x)
print(y)
print(z)
```

Output Variables

- The Python `print` statement is often used to output variables.
- To combine both text and a variable, Python uses the `+` character:

```
x = "awesome"
print("Python is " + x)
```

- You can also use the `+` character to add a variable to another variable:

```
x = "Python is "
y = "awesome"
z =  x + y
print(z)
```

- For numbers, the `+` character works as a mathematical operator:

```
x = 5
y = 10
print(x + y)
```

Output Variables

- If you try to combine a string and a number:

```
x = 5
y = "John"
print(x + y)
```

- Python will give you an error:

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Global Variables

- Variables that are created outside of a function are known as global variables.
- Global variables can be used by everyone, both inside of functions and outside.

```
x = "awesome"

def myfunc():
  print("Python is " + x)

myfunc()
```

Global Variables

- If we create a variable with the same name inside a function, this variable will be local, and can only be used inside the function.
- The global variable with the same name will remain as it was, global and with the original value.

```
x = "awesome"

def myfunc():
  x = "fantastic"
  print("Python is " + x)

myfunc()

print("Python is " + x)
```

The Global Keyword

- To create a global variable inside a function, you can use the `global` keyword.

```
def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)
```

- Also, use the `global` keyword if you want to change a global variable inside a function.

```
x = "awesome"

def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)
```

Thank you