PYTHON PROGRAMMING

Prepared by:

Jasmina Vanasiwala


Python

Python

- Fastest growing language (in terms of)
  - No of developers - who are using it
  - No of libraries we have
  - No of companies - who are using it
  - No of areas you can implement it
    - Machine Learning
    - GUI
    - Software Development
    - Web Development
- Known as – general purpose language

python

- Why famous?
  - Many languages- C, C++, Java
  - Still python famous?
    - Some says – it's new lang X
    - Not exactly – java 1995 & python 1989 – before java
- Easiest language
  - Area – AI/ML – research scientist don't spend much time on prog lang ; they want easiest lang available in market
- Much simpler than C, C++, Java

Companies uses python

- Story ?
  - Name because looks like a snake??
  - The author – big fan of a british comedy movie called- Monty Python's Flying Circus
  - Hence, he named python

Python versions

Python installation

- Python-3.8.1
  - Python interpreter
- IDE-> Integrated Development Environment
  - A place where u write your code
  - Where u'll run it
  - Where u can debug it
  - IDLE, Pycharm
- One IDE that comes by default with python installation is IDLE-
  - Integrated Development Learning Environment
- Pycharm- 2 versions
  - Professional- need to pay some amount to use it
    - provide scientific & web python development
  - Community- free, open source
    - don't provide web development

Note

- u need to insatll python on ur machine- to run python programs
  - u need python interpreter

- once installing python and when you open it-
- that window is called prompt (the window with >>>)

  also when you open IDLE it shows >>> that also called prompt
- in linux we call it as shell

Comments

- used to explain Python code
- used to make the code more readable
- used to prevent execution when testing code

- commenting capability for the purpose of in-code documentation
- Comments start with a #
  - Python will ignore that whole line

```
#This is a comment.
print("Hello, World!")
```

- Comments can be placed at the end of a line, and Python will ignore the rest of the line:

```
print("Hello, World!") #This is a comment
```

```
#print("Hello, World!")
print("Cheers, Mate!")
```

Multi Line Comments

- To add a multiline comment you could insert a # for each line

```
#This is a comment
#written in
#more than just one line
print("Hello, World!")
```

```
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

Variables

- **Name** which is **used to refer memory location**
- Also **known as identifier** and **used to hold value**
- **No need to specify** the **type** of variable
- Python is a type infer (automatic detection of the data type)language
- Variable **names can be a group of both letters and digits**, but they have to **begin with a letter or an underscore**
- Recommended to **use lowercase letters for variable name**
- E.G. **Marks** and **marks** both are **two different variables**

Identifier Naming

- **First character** of the variable **must be** an **alphabet or underscore** (_)
- All the characters except the first character may be an **alphabet of lower-case(a-z)**, **upper-case (A-Z)**, **underscore or digit (0-9)**.
- **Must not contain** any white-space, or **special character** (!, @, #, %, ^, &, *).
- **Must not be similar to any keyword** defined in the language.
- **Case sensitive**
- Examples of valid identifiers : student12, _max, min_5, etc.
- Examples of invalid identifiers: 5a, x%4, a 9, etc.

Assigning Value to Variable

- **equal (=) operator is used to assign value to a variable**

```
name = “jasmina”

marks = 56.76
```

Multiple Assignment

- **Assigning single value to multiple variables**

```
x=y=z=50
```

- **Assigning multiple values to multiple variables**

```
a,b,c=5,10,15
```

Assign Value to Multiple Variables

- allows you to assign values to multiple variables in one line

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

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

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

Python Variables

- In Python, variables are created when you assign a value to it:

- Variables in Python:

```
x = 5
y = "Hello, World!"
```

- Python has no command for declaring a variable

```
x = 5
y = "Hello, World!"
print(x)
print(y)
```

Creating Variables

- containers for storing data values
- no command for declaring a variable
- Variables do not need to be declared with any particular type
- 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)
```

- String variables can be declared either by using single or double quotes:

```
x = "John"
# is the same as
x = 'John'
```

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)
```

- the + character works as a mathematical operator:

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

- If you try to combine a string and a number, Python will give you an error:

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

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

Getting the Data Type

- get the data type of any object by using the type() function:

- Example- Print the data type of the variable x:

```
x = 5
print(type(x))
```

Output- <class 'int'>

Setting the Data Type

In Python, the data type is set when you assign a value to a variable:

Example	Data Type
`x = "Hello World"`	str
`x = 20`	int
`x = 20.5`	float
`x = 1j`	complex
`x = ["apple", "banana", "cherry"]`	list
`x = ("apple", "banana", "cherry")`	tuple
`x = range(6)`	range
`x = {"name" : "John", "age" : 36}`	dict
`x = {"apple", "banana", "cherry"}`	set

Setting the Specific Data Type

If you want to specify the data type, you can use the following constructor functions:

Example	Data Type
`x = str("Hello World")`	str
`x = int(20)`	int
`x = float(20.5)`	float
`x = complex(1j)`	complex
`x = list(("apple", "banana", "cherry"))`	list
`x = tuple(("apple", "banana", "cherry"))`	tuple
`x = range(6)`	range
`x = dict(name="John", age=36)`	dict
`x = set(("apple", "banana", "cherry"))`	set

Pythin numbers

- There are three numeric types in Python:
  - int
  - float
  - complex

- Variables of numeric types are created when you assign a value to them

- To verify the type of any object in Python, use the type() function

Note:

Please refer examples attached for the same

Python Numbers

- Complex number
- **Description**
- A complex number is a number that can be expressed in the form a + bi, where a and b are real numbers, and i is a solution of the equation $x^2 = -1$. Because no real number satisfies this equation, i is called an imaginary number.

Type Conversion

- You can convert from one type to another with the int(), float(), and complex() methods

- You cannot convert complex numbers into another number type.

Note:

Please refer examples attached for the same

Specify a Variable Type

- There may be times when you want to specify a type on to a variable.
- This can be done with casting.

- int() - constructs an integer number from an integer literal, a float literal (by rounding down to the previous whole number), or a string literal (providing the string represents a whole number)

- float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)

- str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals

Strings

- Strings can be **created by enclosing the character** or the **sequence of characters in the quotes**.
- Python **allows** to use **single quotes**, **double quotes**, or **triple quotes** to create the string.
- Example
  - text = "Hello there!"
- If we check the type of the variable text using a python script
  - print(type(text)) => then it will print string (str).
- **Strings are treated as the sequence of strings** which means that **python doesn't support the character data type** instead a **single character written as 'r' is treated as the string of length 1**.

You can display a string literal with the print() function:

Example
print("Hello")
print('Hello')

a = "Hello"
print(a)

Multiline Strings

- assign a multiline string to a variable by using three quotes (single/double)

```
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)
```

```
a = '''Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.'''
print(a)
```

the line breaks are inserted at the same position as in the code

Strings are Arrays

- Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters

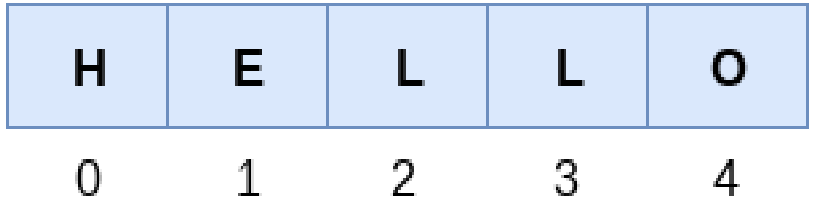
- However, Python does not have a character data type, a single character is simply a string with a length of 1

- Square brackets can be used to access elements of the string

Strings indexing and splitting

- **Indexing** of the python strings **starts from 0**.
- Example, The string "HELLO" is indexed as given in the figure.
- the **slice operator [ ]** is used to access the individual characters of the string.

str = "HELLO"



str[0] = 'H'

str[1] = 'E'

str[2] = 'L'

str[3] = 'L'

str[4] = 'O'

- Get the character at position 1 (remember that the first character has the position 0):

```
a = "Hello, World!"
print(a[1])
```

Slicing

- You can return a range of characters by using the slice syntax
- Specify the start index and the end index, separated by a colon, to return a part of the string

Get the characters from position 2 to position 5 (not included):

```
b = "Hello, World!“
print(b[2:8])
```

Answer- llo, W

- We can use the **: (colon) operator** in python to **access the substring**.

str = "HELLO"

str[0] = 'H'

str[1] = 'E'

str[2] = 'L'

str[3] = 'L'

str[4] = 'O'

str[:] = 'HELLO'

str[0:] = 'HELLO'

str[:5] = 'HELLO'

str[:3] = 'HEL'

str[0:2] = 'HE'

str[1:4] = 'ELL'

Negative Indexing

- Get the characters from position 5 to position 1, starting the count from the end of the string:

```
b = "Hello, World!"
print(b[-5:-2])
```

Output- orl

Escape Character

- To insert characters that are illegal in a string, use an escape character.
- An escape character is a backslash \ followed by the character you want to insert.
- An example of an illegal character is a double quote inside a string that is surrounded by double quotes:
- You will get an error of invalid syntax (syntax error) if you use double quotes inside a string that is surrounded by double quotes:

```
txt = "We are the so-called "Vikings" from the north."
```

- To fix this problem, use the escape character \":
- Example:

The escape character allows you to use double quotes when you normally would not be allowed:

```
txt = "We are the so-called \"Vikings\" from the north.“
```

Output- We are the so-called "Vikings" from the north.

Note

- You can’t replace/update any particular character at certain index in your string directly- since item assignment isn’t possible in python
- You can’t insert a character at certain index in your string directly
- You can’t remove particular character from certain index directly
- You must have to write logic for the same

String Operators

Operator	Description
+	It is known as concatenation operator used to join the strings given either side of the operator.
	str = "Hello" str1 = " world" print(str+str1)# prints Hello world
*	It is known as repetition operator. It concatenates the multiple copies of the same string.
	str = "Hello" str1 = " world" print(str*3) # prints HelloHelloHello
[]	It is known as slice operator. It is used to access the sub-strings of a particular string.
	str = "Hello" print(str[4]) # prints o

Operator	Description
[:]	It is known as range slice operator. It is used to access the characters from the specified range.
	str = "Hello" print(str[2:4]); # prints ll
in	It is known as membership operator. It returns if a particular sub-string is present in the specified string.
	str = "Hello" print('w' in str) # prints false as w is not present in str
not in	It is also a membership operator and does the exact reverse of in. It returns true if a particular substring is not present in the specified string.
	str1 = " world" print('wo' not in str1) # prints false as wo is present in str1.

Operator	Description
r/R	It is used to specify the raw string. Raw strings are used in the cases where we need to print the actual meaning of escape characters such as "C://python". To define any string as a raw string, the character r or R is followed by the string.
	print(r'C://python37') # prints C://python37 as it is written
%	It is used to perform string formatting. It makes use of the format specifiers used in C programming like %d or %f to map their values in python.
	str = "Hello" print("The string str : %s"%(str)) # prints The string str : Hello

String Format

txt = "My name is John, I am " + age
TypeError: must be str, not int

But we can combine strings and numbers by using the `format()` method!

The `format()` method takes the passed arguments, formats them, and places them in the string where the placeholders `{}` are:

Use the format() method to insert numbers into strings:

```
age = 36
txt = "My name is John, and I am {}"
print(txt.format(age))
```

The format() method takes unlimited number of arguments, and are placed into the respective placeholders:

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price))
```

I want 3 pieces of item 567 for 49.95 dollars.

- use index numbers {0} to be sure the arguments are placed in the correct placeholders:

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))
```

I want to pay 49.95 dollars for 3 pieces of item 567

- To make sure a string will display as expected, we can format the result with the format() method.
- allows you to format selected parts of a string.
- The format() method returns the formatted string.
- Sometimes there are parts of a text that you do not control, maybe they come from a database, or user input?
- To control such values, add placeholders (curly brackets {}) in the text, and run the values through the format() method:

- Add a placeholder where you want to display the price:

```
price = 49
txt = "The price is {} dollars"
print(txt.format(price))
```

- You can add parameters inside the curly brackets to specify how to convert the value:

- Format the price to be displayed as a number with two decimals:

```
price = 49
txt = "The price is {:.2f} dollars"
print(txt.format(price))
```

```
The price is 49.00 dollars
```

```
txt = "For only {price:.2f} dollars!"
print(txt.format(price = 49))
```

```
For only 49.00 dollars!
```

```
quantity = 3
itemno = 567
price = 49
myorder = "I want {0} pieces of item number {1} for {2:.2f} dollars."
print(myorder.format(quantity, itemno, price))
```

```
I want 3 pieces of item number
567 for 49.00 dollars.
```

Named Indexes

use named indexes by entering a name inside the curly brackets {carname},

- you must use names when you pass the parameter values txt.format(carname = "Ford"):

```
myorder = "I have a {carname}, it is a {model}.“

print(myorder.format(carname = "Ford", model = "Mustang"))
```

Python Formatting operator

- Python allows us to use the format specifiers used in C's printf statement.
- provides an **additional operator %**
  - used as an interface between the format specifiers and their values.

- binds the format specifiers to the values.

```
Integer = 10
Float = 1.290
String = "Ayush"
print("Hi I am Integer ... My value is %d\nHi I am float ... My value is %f\nHi I am string ... My value is %s"%(Integer,Float,String));
```

Output:

```
Hi I am Integer ... My value is 10
Hi I am float ... My value is 1.290000
Hi I am string ... My value is Ayush
```

- Python 3.6 added a new string formatting approach called formatted string literals or “f-strings”

- This new way of formatting strings lets you use embedded Python expressions inside string constants

- this prefixes the string constant with the letter “f“
  —hence the name “f-strings”

- even do inline arithmetic with it

Python Conditions and If statements

Python supports the usual logical conditions from mathematics:

- Equals: `a == b`
- Not Equals: `a != b`
- Less than: `a < b`
- Less than or equal to: `a <= b`
- Greater than: `a > b`
- Greater than or equal to: `a >= b`

These conditions can be used in several ways, most commonly in "if statements" and loops.

An "if statement" is written by using the `if` keyword.

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.

```
while expression:
        statement(s)
```

- The while loop requires an indexing variable to set beforehand.

While …

- The break Statement
  - With the break statement we can stop the loop even if the while condition is true
- The continue Statement
  - With the continue statement we can stop the current iteration, and continue with the next

While

- The else Statement
  - With the else statement we can run a block of code once when the condition no longer is true:

Python For Loops

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

- 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.

```
for iterator_var in sequence:
    statements(s)
```

For ...

- The break Statement
  - With the break statement we can stop the loop before it has looped through all the items
- The continue Statement
  - With the continue statement we can stop the current iteration of the loop, and continue with the next

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.

Else in For Loop

- The else keyword in a for loop specifies a block of code to be executed when the loop is finished

Nested Loops

- A nested loop is a loop inside a loop

Python Booleans

- Booleans represent one of two values:
  True or False

- Boolean Values
  - In programming you often need to know if an expression is True or False
  - You can evaluate any expression in Python, and get one of two answers, True or False
  - When you compare two values, the expression is evaluated and Python returns the Boolean answer
  - print(10 > 9)
  - print(10 == 9)
  - print(10 < 9)

- When you run a condition in an if statement, Python returns True or False

```
a = 200
b = 33

if b > a:
        print("b is greater than a")
else:
        print("b is not greater than a")
```

- Evaluate Values and Variables
  - The bool() function allows you to evaluate any value, and give you True or False in return
  - print(bool("Hello"))
    print(bool(15))
  - x = "Hello"
    y = 15

    print(bool(x))
    print(bool(y))

- Most Values are True
  - Almost any value is evaluated to True if it has some sort of content.
  - Any string is True, except empty strings.
  - Any number is True, except 0.
  - Any list, tuple, set, and dictionary are True, except empty ones.
  - bool("abc")
    bool(123)
    bool(["apple", "cherry", "banana"])

- Some Values are False

```
bool(False)
bool(None)
bool(0)
bool("")
bool(())
bool([])
bool({})
```

- Functions can Return a Boolean
  - built-in function that returns a boolean value,
  - like the **isinstance()** function,
  - which can be used to determine if an object is of a certain data type
  - Check if an object is an integer or not:
    - x = 200
      print(isinstance(x,int))

Python Collections (Arrays)

- There are four collection data types in the Python programming language:
  - **List** is a collection which is ordered and changeable. Allows duplicate members.
  - **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
  - **Set** is a collection which is unordered and unindexed. No duplicate members.
  - **Dictionary** is a collection which is unordered, changeable and indexed. No duplicate members.

List

- A list is a collection which is ordered and changeable.
- In Python lists are written with square brackets.
- thislist = ["apple", "banana", "cherry"]
  print(thislist)

- Copy a List
  - You cannot copy a list simply by typing list2 = list1,
  - because: list2 will only be a reference to list1, and changes made in list1 will automatically also be made in list2.
  - There are ways to make a copy, one way is to use the built-in List method copy().

Tuple

- A tuple is a collection which is ordered and **unchangeable**.
- In Python tuples are written with round brackets.
- thistuple = ("apple", "banana", "cherry")
  print(thistuple)

- Add Items
  - Once a tuple is created, you cannot add items to it.
  - Tuples are **unchangeable**.
  - thistuple = ("apple", "banana", "cherry")
    thistuple[3] = "orange" # This will raise an error
    print(thistuple)
  - TypeError: 'tuple' object does not support item assignment

- Change Tuple Values
  - Once a tuple is created, you cannot change its values.
  - Tuples are **unchangeable**, or **immutable**
  - convert the tuple into a list, change the list, and convert the list back into a tuple.

Create Tuple With One Item

- thistuple = ("apple",)
  print(type(thistuple))

  #NOT a tuple
  thistuple = ("apple")
  print(type(thistuple))

- Output- <class 'tuple'>
  <class 'str'>

Remove Items

- Tuples are **unchangeable** - you cannot remove items from it
- you can delete the tuple completely
- del keyword

Example- thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple) #this will raise an error because the tuple no longer exists

Python Sets

- A set is a collection which is unordered and unindexed.
- In Python sets are written with curly brackets.

Access Items

- You cannot access items in a set by referring to an index, since sets are unordered the items has no index.

- But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

Remove Item

- use the remove(), or the discard() method

- thisset = {"apple", "banana", "cherry"}
  thisset.remove("banana")
  print(thisset)

- If the item to remove does not exist, remove() will raise an error.

- thisset = {"apple", "banana", "cherry"}
  thisset.discard("banana")
  print(thisset)

- If the item to remove does not exist, discard() will NOT raise an error.

Remove Item

- use pop() method to remove an item - remove the last item
- sets are unordered, so you will not know what item that gets removed

- thisset = {"apple", "banana", "cherry"}
  x = thisset.pop()
  print(x) #removed item
  print(thisset) #the set after removal

- apple
  {'banana', 'cherry'}

clear() method

- The clear() method empties the set:

- thisset = {"apple", "banana", "cherry"}
  thisset.clear()
  print(thisset)

- Output- set()

del() keyword

- The del keyword will delete the set completely:

- thisset = {"apple", "banana", "cherry"}
  del thisset
  print(thisset)

- Output- NameError: name 'thisset' is not defined

Join Two Sets

- union() method

- set1 = {"a", "b" , "c"}
  set2 = {1, 2, 3}
  set3 = set1.union(set2)
  print(set3)

- Output- {3, 'b', 2, 1, 'a', 'c'}

Join Two Sets

- update() method

- set1 = {"a", "b" , "c"}
  set2 = {1, 2, 3}
  set1.update(set2)
  print(set1)

- Output- {3, 'c', 'a', 'b', 1, 2}

- Both union() and update() will exclude any duplicate items

The set() Constructor

- set() constructor to make a set

- thisset = set(("apple", "banana", "cherry"))
  print(thisset)

Note: the set list is unordered, so the result will display the items in a random order.

- Output- {'apple‘, 'cherry', 'banana'}

Python Dictionaries

- A dictionary is a collection which is unordered, changeable and indexed.
- In Python dictionaries are written with curly brackets, and they have keys and values.

Check if Key Exists

- thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
  }
  if "model" in thisdict:
  print("Yes, 'model' is one of the keys in the thisdict dictionary")

Dictionary Length

- To determine how many items (key-value pairs) a dictionary has,
- use the len() method

- thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
  }
  print(len(thisdict))

output- 3

Copy a Dictionary

- copy() method

- ```
  thisdict = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
  }
  mydict = thisdict.copy()
  print(mydict)
  ```

- Output- {'brand': 'Ford', 'model': 'Mustang', 'year': 1964}

Note:

You cannot copy a dictionary simply by typing dict2 = dict1, because: dict2 will only be a reference to dict1, and changes made in dict1 will automatically also be made in dict2.
  ```

Copy a Dictionary …

- Make a copy of a dictionary with the dict() method:

- ```
  thisdict = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
  }
  mydict = dict(thisdict)
  print(mydict)
  ```
  ```

dict() Constructor

```
thisdict = dict(brand="Ford", model="Mustang", year=1964)

# note that keywords are not string literals
# note the use of equals rather than colon for the assignment

print(thisdict)
```

Nested Dictionaries

- A dictionary can also contain many dictionaries, this is called nested dictionaries.

- Create a dictionary that contain three dictionaries:

- ```
  myfamily = {
    "child1" : {
      "name" : "Emil",
      "year" : 2004
    },
    "child2" : {
      "name" : "Tobias",
      "year" : 2007
    },
    "child3" : {
      "name" : "Linus",
      "year" : 2011
    }
  }
  print(myfamily)
  ```

Output-

{'child1': {'name': 'Emil', 'year': 2004},
'child2': {'name': 'Tobias', 'year': 2007},
'child3': {'name': 'Linus', 'year': 2011}}
  ```

- If you want to nest three dictionaries that already exists as dictionaries:

```
child1 = {
  "name" : "Emil",
  "year" : 2004
}
child2 = {
  "name" : "Tobias",
  "year" : 2007
}
child3 = {
  "name" : "Linus",
  "year" : 2011
}

myfamily = {
  "child1" : child1,
  "child2" : child2,
  "child3" : child3
}
```

Function

- 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.

- Creating a Function:
  - In Python a function is defined using the def keyword:

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

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

Output- Hello from 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 are sent to the function when it is called.

- def my_function(**fname**):
  print(fname + " Martin")

  my_function("**Emil**")
  my_function("**Tobias**")
  my_function("**Linus**")

Output-

Emil Martin
Tobias Martin
Linus Martin

- def my_function(fname, lname):
  print(fname + " " + lname)

  my_function("Emil", “Martin") Output- Emil Martin

- In below function: expects 2 arguments, but gets only 1

- def my_function(fname, lname):
  print(fname + " " + lname)

  my_function("Emil")

- Output- TypeError: my_function() missing 1 required positional argument: 'lname'

Keyword Arguments

- You can also send arguments with the *key = value* syntax.
- This way the order of the arguments does not matter.
- def my_function(child3, child2, child1):
  print("The youngest child is " + child3)

  my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")

Default Parameter Value

- If we call the function without argument, it uses the default value:

- def my_function(**country = "Norway"**):
  print("I am from " + country)

  my_function("Sweden")
  my_function("India")
  my_function()
  my_function("Brazil")

- def my_function(food):
  for x in food:
    print(x)

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

  my_function(fruits)

Note:

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.

Passing a List as an Argument
E.g. if you send a List as an argument, it will still be a List when it reaches the function:

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))
  print(my_function(9))

Python Scope

- Scope- A variable is only available from inside the region it is created
    - Local Scope
    - Global Scope

Local Scope

- A variable created inside a function belongs to the *local scope* of that function,
- can only be used/available inside that function

Function Inside Function

- in the example above, the variable x is not available outside the function, but - available for any function inside the function

Global Scope

- created outside of a function

  global variable - belongs to the global scope

- Global variables can be used by everyone,

  both inside of functions and outside

Naming Variables

- If you operate with the same variable name inside and outside of a function,
  - Python will treat them as two separate variables,
  - one available in the global scope (outside the function)
  - one available in the local scope (inside the function)

- Example- Create a variable inside a function, with the same name as the global variable

```
x = "awesome“    #global

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

myfunc()

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

```
Python is fantastic
Python is awesome
```

Note:

If you 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.

The global Keyword

- The global keyword makes the variable global
- To create a global variable inside a function, you can use the global keyword

- Example- If you use the global keyword, the variable belongs to the global scope:

```
def myfunc():
  global x
  x = 300

myfunc()

print(x)
```

- use the global keyword if you want to change a global variable inside a function

- Example- To change the value of a global variable inside a function, refer to the variable by using the global keyword:

```
x = 300

def myfunc():
  global x
  x = 200

myfunc()

print(x)
```

```
200
```

```
x = 300

def myfunc():
  x = 200
  print(x)

myfunc()

print(x)
```

```
200
300
```

Python User Input

- ask the user for input

- Python 3.6 uses the input() method
- Python 2.7 uses the raw_input() method

- Python stops executing when it comes to the input() function,
  - and continues when the user has given some input

- Anything given to input is returned as a string.
- So, if we give an integer like 5, we will get a string i.e. '5' (a string) and not 5 (int).
- to take integer input from the user
  - you have to explicitly convert to int
- You can accept numbers of any base
  - convert them directly to base-10 with the int function

Thank You!