# Introduction To Python

Hands-on Training

**Duration:** 1 Day, Level: Introductory

**Dr. Vijay Nagarajan PhD**

American Academy For Biomedical Informatics, USA

# Table Of Contents

# 1. A simple Python program

## 1.1. The Hello World program

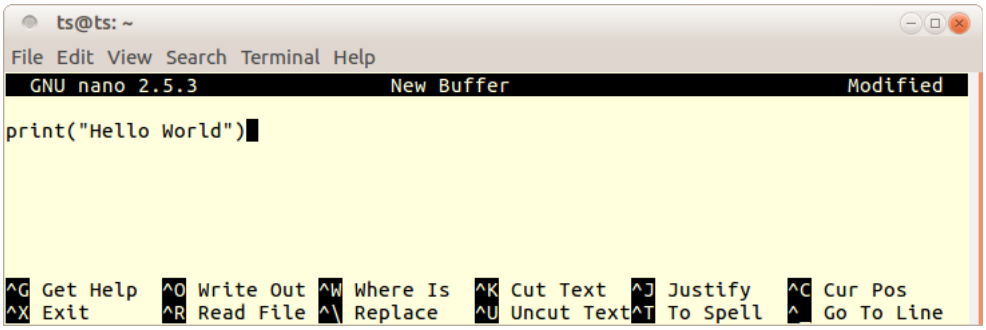
1.1.1. Use the “python3 -V” command to see what version of python we have;

```
ts@ts:~$ python3 -V
Python 3.5.2
ts@ts:~$
```

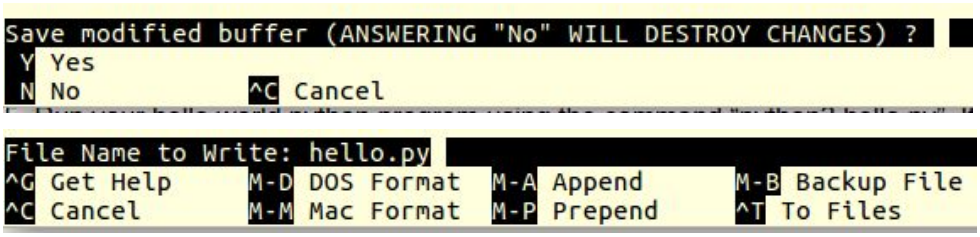
1.1.2. Open a text editor (nano) using the “nano” command;

```
ts@ts:~$ nano
ts@ts:~$
```

1.1.3. Type the command `print("Hello World")` in the nano editor;




1.1.4. Press the Ctrl+X keys to “Exit” nano, when prompted press “Y” and enter the file name “hello.py”, to save the program as a python script file. List the files to make sure you have the “hello.py” in your working directory;




## 1.2. Running the Hello World program

1.2.1. Run your hello world python program using the command “python3 hello.py”. If the program runs successfully, the interpreter would print “hello world” to the standard output (screen);

```
ts@ts:~$ python3 hello.py
Hello World
ts@ts:~$
```

## 1.3. Comments

1.3.1. Open the ‘hello.py’ program using the command ‘nano hello.py’, add a single line comment “# This is my first program”. Save the program and run it as we did above;

```
  GNU nano 2.5.3                File: hello.py

#This is my first python program
print("Hello World")
```

## 1.4. Variables

1.4.1. Open nano, write the following program and save the file as “conditions.py”;

```
#This program introduces variables, operators and conditions
name="Raj"
age=22
print("My Name is:",name)
print("My Age is:",age)
```

1.4.2. Running “conditions.py” program should produce the following output;

```
ts@ts:~/py$ python3 conditions.py
My Name is: Raj
My Age is: 22
ts@ts:~/py$
```

## 1.5. Operators

1.5.1. Modify the ‘conditions.py’ script, so that it has the contents like below (assignment operator “=”, subtraction operator “-”);

```
#This program introduces variables, operators and conditions
name="Raj"
age=22

brothername="Vijay"
brotherage=26

agedifference=age-brotherage
difference=abs(agedifference)

print("My Name is:",name)
print("My Age is:",age)
print("My Brothers Name is:", brothername)
print("My Brothers Age is:", brotherage)
print("We were born",difference,"years apart")
```

## 1.6. Statements

In the above code (1.5.1), the assignment operation age=22 is called an assignment statement. This statement does not return any values. The interpreter simply executes this statement.

## 1.7. Expressions

In the above code (1.5.1), the line “age-brotherage” is an expression, which is evaluated by the interpreter, which then outputs the resulting value, which is then assigned to a variable. Likewise the ‘print’ lines are also examples of expressions.

# 2. Conditionals

## 2.1. “if” statement

2.1.1. Modify the conditions.py code, using the ‘if’ conditional statement, to check if “Vijay” is older than “Raj” and print the result, like given below;

```
#This program introduces variables, operators and conditions
name="Raj"
age=22

brothername="Vijay"
brotherage=26

agedifference=age-brotherage
difference=abs(agedifference)

print("My Name is:",name)
print("My Age is:",age)
print("My Brothers Name is:", brothername)
print("My Brothers Age is:", brotherage)
print("We were born",difference,"years apart")

if brotherage > age:
        print("My brother",brothername,"is older than me")
```

## 2.2. “if” “else” statement

2.2.1. Modify the ‘if’ statement in the previous conditions.py code, using the ‘if’ ‘else’ conditional statement, to check who is the older brother and print the result, like given below;

```
if brotherage < age:
        print("I am older than my brother",brothername)
else:
        print("My brother",brothername,"is older than me")
```

2.2.2. Result of 2.2.1;

```
kce1@bl8vbox[kce1] python3 conditions.py
My Name is: Raj
My Age is: 22
My Brothers Name is: Vijay
My Brothers Age is: 26
We were born 4 years apart
My brother Vijay is_older than me
```

# 3. Loops

## 3.1. “for” loop

3.1.1. Use nano to code the following ‘for’ loop example (printing the numbers 1 to 9), save the script as loops.py and print out the results;

```
  GNU nano 2.2.6                                File: loops.py

#This example prints the first 1 to 10 numbers

for i in range(1,10):
        print(i)
```

3.1.2. Result of 3.1.1;

```
kce1@bl8vbox[kce1] python3 loops.py
1
2
3
4
5
6
7
8
9
```

3.1.3. Modify the loops.py code (like shown below), to print out the 2 times table;

```
#This example prints the first 1 to 10 numbers

for i in range(1,11):
        print(i,"x 2 =",i*2)
```

3.1.4. Result of 3.1.3;

```
kce1@bl8vbox[kce1] python3 loops.py
1 x 2 = 2
2 x 2 = 4
3 x 2 = 6
4 x 2 = 8
5 x 2 = 10
6 x 2 = 12
7 x 2 = 14
8 x 2 = 16
9 x 2 = 18
10 x 2 = 20
```

## 3.2. “while” loop

3.2.1. Add the “while” loop statement in the loops.py script, like shown below and output the results;

```
#This example prints the first 1 to 10 numbers

for i in range(1,11):
        print(i,"x 2 =",i*2)

#This example prints all the numbers that are less than 10

mynumber=0
while mynumber < 10:
        print(mynumber)
        mynumber=mynumber+1
```

The while loop will execute the statements within the loop, as long as the while condition is true.

# 4. Lists

## 4.1. Creating a list

4.1.1. Using nano, create a list of programming languages and print the list items, using the below code (save the script as lists.py);

```
  GNU nano 2.2.6                                File: lists.py

#This is an example python script with lists

lang=["C","C++","Java","Python","PHP","Perl"]
print(lang)
```

## 4.2. Accessing list elements

4.2.1. Modify the lists.py script, as shown below, to access a specific element using its index;

```
#This is an example python script with lists

lang=["C","C++","Java","Python","PHP","Perl"]
print(lang)

#Print the second element from the list
print(lang[1])
```

4.2.2. Result of 4.2.1 given below;

```
kce1@bl8vbox[kce1] python3 lists.py
['C', 'C++', 'Java', 'Python', 'PHP', 'Perl']
C++
```

4.2.3. Modify the lists.py script, as shown below, to check if a specific element is present in a list and to print that elements index position;

```
#This is an example python script with lists

lang=["C","C++","Java","Python","PHP","Perl"]
print(lang)

#Print the second element from the list
print(lang[1])

#Search the list for an item
if "Java" in lang:
        print(lang.index("Java"))
```

4.2.4. Result of 4.2.3, shown below;

```
kce1@bl8vbox[kce1] python3 lists.py
['C', 'C++', 'Java', 'Python', 'PHP', 'Perl']
C++
2
```

## 4.3. Adding an element to the list

4.3.1. Modify the lists.py script to include the following code, to add a new item in the programming languages list;

```
#This is an example python script with lists

lang=["C","C++","Java","Python","PHP","Perl"]
print(lang)

#Print the second element from the list
print(lang[1])

#Search the list for an item
if "Java" in lang:
        print(lang.index("Java"))

#Appending an item to the list
lang.append("Scala")
print(lang)
```

4.3.2. Results of 4.3.1 shown below;

```
kce1@bl8vbox[kce1] python3 lists.py
['C', 'C++', 'Java', 'Python', 'PHP', 'Perl']
C++
2
['C', 'C++', 'Java', 'Python', 'PHP', 'Perl', 'Scala']
```

## 4.4. Deleting a list element

4.4.1. Modify the lists.py code like shown below, to delete “Perl” from the list;

```
#This is an example python script with lists

lang=["C","C++","Java","Python","PHP","Perl"]
print(lang)

#Print the second element from the list
print(lang[1])

#Search the list for an item
if "Java" in lang:
        print(lang.index("Java"))

#Appending an item to the list
lang.append("Scala")
print(lang)

#Delete an item from the list
del(lang[5])
print(lang)
```

## 4.5. Slicing the list

4.5.1. Extract (slice) part of the list, from the lists.py script, by including the following code in that script;

```
#Slicing the list
print(lang[2:])
print(lang[:3])
print(lang[1:3])
```

## 4.6. Iterating through the list

4.6.1. Use the ‘for’ loop, as shown in the below code, to iterate through and print each of the items in the programming languages list, in the lists.py script;

```
#Iterating through the list
for language in lang:
        print(language)
```

# 5. Tuples

Tuples are immutable sequence objects, whereas lists are mutable.

## 5.1. Creating a tuple

5.1.1. Use nano to write a small script as shown below, with tuples of software companies and print out the tuples. Save the script as tuples.py;

```
  GNU nano 2.2.6                                File: tuples.py

#This programs shows a tuples example

companies=("Apple","Microsoft","Google")
print(companies)
```

Tuples items are accessed the same way as list items. Similar operations could be done in lists and tuples.

# 6. Dictionaries

## 6.1. Creating a dictionary

6.1.1. A dictionary in python is a list of key-value associated items. Use nano to create a script called dict.py, with the following code;

```
#This program demonstrates dictionaries in python
dict={'Home':'DC', 'Language':'English', 'Car':'Black','Food':'Burger'}
print(dict['Language'])
```

Several operations that are done in lists and tuples could also be applied for dictionaries.

# 7. Functions

Functions are a named (defined) block of code that could be reused in a program.

## 7.1. Writing and Calling a Function

7.1.1. Use nano to write the following code, with a function (‘addtwo’) defined to add two numbers and return the total. This script also gets the input from the user. Save the script as function.py.

```
  GNU nano 2.2.6                File: function.py

#This program demonstrates functions in python

def addtwo(x,y):
        total=int(x)+int(y)
        print(total)
        return

#User input
firstn=input('Enter first number:')
secondn=input('Enter second number:')

#Calling the function
addtwo(firstn,secondn)
```

# 8. Reading a file

## 8.1. Create a file

Create a file using nano, with the following contents (save the file as names.tx);

```
  GNU nano 2.2.6                File: names.txt

1       CSE
2       EEE
3       ECE
4       ETE
```

## 8.2. Read the entire file content

Write a script (read.py) with the following code, to read the names.txt file and dump its contents to the screen;

```
  GNU nano 2.2.6                File: read.py

#This script demonstrates reading a file in python
contents=open("names.txt","r")
print(contents.read())
```

## 8.3. Read the file content line-by-line

Modify the read.py script with the below code, to read the names.txt file line-by-line and print the contents;

```
  GNU nano 2.2.6                File: read.py

#This script demonstrates reading a file in python
contents=open("names.txt","r")
#print(contents.read())
for line in contents:
        print(line)
```

# 9. Writing to a file

Modify the read.py script with the below code, to write contents to the names.txt file;
Check out the names.txt after running this read.py script.

```
  GNU nano 2.2.6                File: read.py

#This script demonstrates reading a file in python
contents=open("names.txt","r+")
#print(contents.read())
for line in contents:
        print(line)

contents.write("BTech\n")
contents.write("MTech\n")
print(contents.read())
```

# 10. Testing

## 10.1. doctest

Write the following script (test.py), describing a function that squares the given number x. The function is tested using the doctest module of python. The docstring within the function defines the testing to be carried out.

```
  GNU nano 2.2.6                File: test.py

def square(x):
    """
    >>> square(4)
    16
    >>> square(5)
    25
    """
    return x*2
#########################
if __name__ == "__main__":
    import doctest
    doctest.testmod(verbose=True)
```

Fix the script, so that both the tests pass.

# 11. Debugging

Write the script debug.py with the following code, containing the ‘import pdb’ - the default python debugger, with the trace starting from the beginning of the script;

```
  GNU nano 2.2.6                File: debug.py

#This script demonstrates the use of pdb
import pdb

#Start tracing here
pdb.set_trace()
one=1
two=2
three=3

total=one+two+three
print(total)
```

## 11.1. Tracing

Running the debug.py script would start tracing with the pdb prompt, as seen below;

```
kce1@bl8vbox[kce1] python3 debug.py
> /home/kce1/debug.py(6)<module>()
-> one=1
(Pdb)
```

11.1.1. Use ‘n’ and ENTER to skip to the next line of code. Use ‘p’ and the variable name to print the value of that variable. Use ‘q’ to quit the debugger.

----------END----------

vijay@graphinformatics.com