**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**](#_8qdhf1po657c) **3**

[1.1. The Hello World program](#_5fa4t58xqr5u) 3

[1.2. Running the Hello World program](#_j0261aoby91g) 3

[1.3. Comments](#_82a4lm2aagds) 4

[1.4. Variables](#_w33oce1hav4b) 4

[1.5. Operators](#_3m55z9mnp6fd) 4

[1.6. Statements](#_p6lk1o8kijhx) 5

[1.7. Expressions](#_2i5ltt7uha9x) 5

[**2. Conditionals**](#_6mh8v56otyt0) **5**

[2.1. “if” statement](#_8ln7v0p7g6mp) 5

[2.2. “if” “else” statement](#_80op3k3xs64o) 5

[**3. Loops**](#_n6a3rpkmnze8) **6**

[3.1. “for” loop](#_avopymu9xh1e) 6

[3.2. “while” loop](#_am26ulgk0mw) 7

[**4. Lists**](#_5bax6nayn687) **7**

[4.1. Creating a list](#_yqvrrw2keku6) 7

[4.2. Accessing list elements](#_7afbqor8tp40) 8

[4.3. Adding an element to the list](#_386i6zuvqbvh) 8

[4.4. Deleting a list element](#_8bp670yjaqp5) 9

[4.5. Slicing the list](#_vm875knypm29) 10

[4.6. Iterating through the list](#_7uaphfljzhni) 10

[**5. Tuples**](#_djj2r6zaj0ds) **10**

[5.1. Creating a tuple](#_9m55lxef2gag) 10

[**6. Dictionaries**](#_jqbddrqip5bz) **11**

[6.1. Creating a dictionary](#_guy3d5r0sdkk) 11

[**7. Functions**](#_e28z16oiv30p) **11**

[7.1. Writing and Calling a Function](#_flw7u8fkjiif) 11

[**8. Reading a file**](#_tvgnatr6ildk) **12**

[8.1. Create a file](#_sgj4btwdpq8x) 12

[8.2. Read the entire file content](#_fqnkb2wt50u2) 12

[8.3. Read the file content line-by-line](#_jvriai4aqjx8) 12

[**9. Writing to a file**](#_iriqfjrznwix) **12**

[**10. Testing**](#_6lx4hcvdldq9) **13**

[10.1. doctest](#_ukgppitzb9ki) 13

[**11. Debugging**](#_27dvw61de07u) **13**

[11.1. Tracing](#_xi6wf5ef2h77) 14

# 1. A simple Python program

## 1.1. The Hello World program

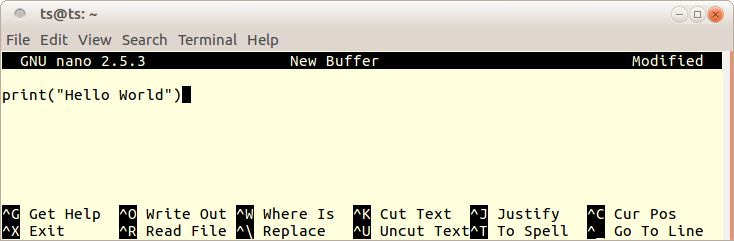
1.1.1. Use the “python3 -V” command to see what version of python we have;



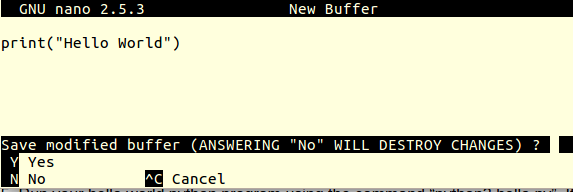
1.1.2. Open a text editor (nano) using the “nano” command;



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;





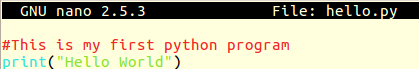
## 1.2. Running the Hello World program

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);



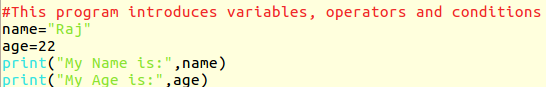
## 1.3. Comments

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;

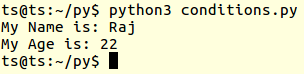


## 1.4. Variables

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

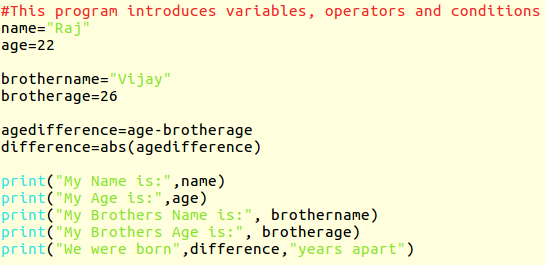


1.4.2. Running “conditions.py” program should produce the following output;



## 1.5. Operators

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



## 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.

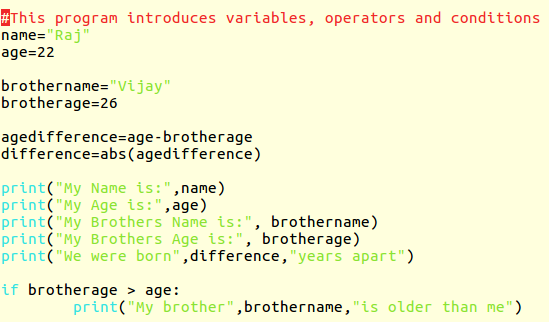
## 1.7. Expressions

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

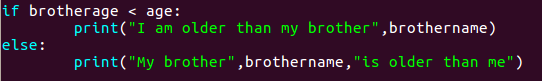
## 2.1. “if” statement

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;

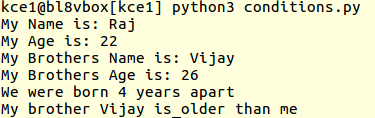


## 2.2. “if” “else” statement

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;



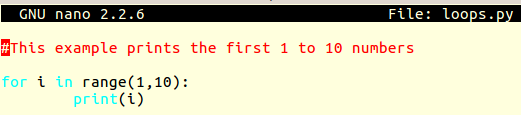
2.2.2. Result of 2.2.1;



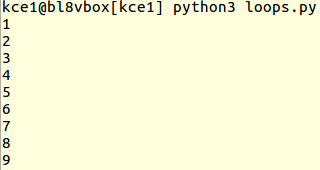
# 3. Loops

## 3.1. “for” loop

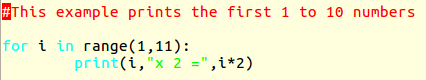
3.1.1. Use nano to code the following ‘for’ loop example (printing the numbers 1 to 9), save the script as loops.py and print out the results;



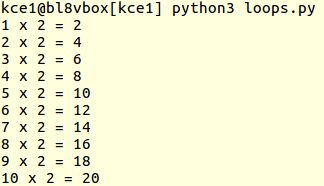
3.1.2. Result of 3.1.1;



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

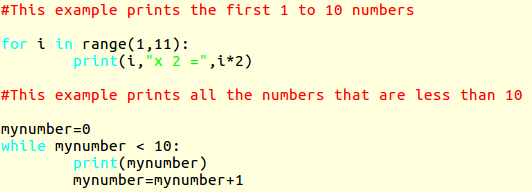


3.1.4. Result of 3.1.3;



## 3.2. “while” loop

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

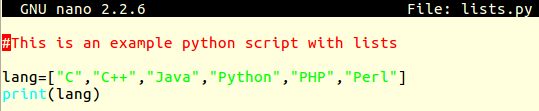


The while loop will execute the statements within the loop, as long as the while condition is true.

# 4. Lists

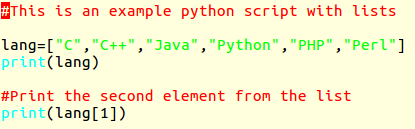
## 4.1. Creating a list

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);



## 4.2. Accessing list elements

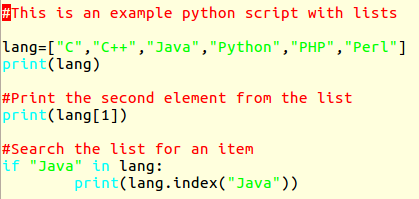
4.2.1. Modify the lists.py script, as shown below, to access a specific element using its index;



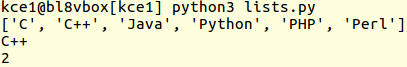
4.2.2. Result of 4.2.1 given below;



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;

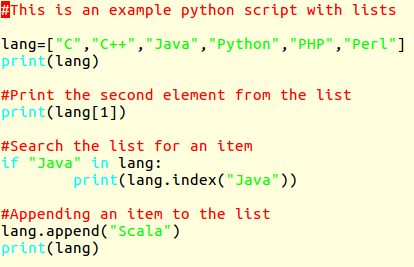


4.2.4. Result of 4.2.3, shown below;

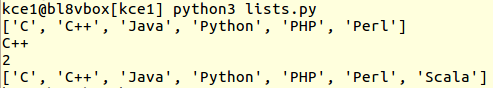


## 4.3. Adding an element to the list

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

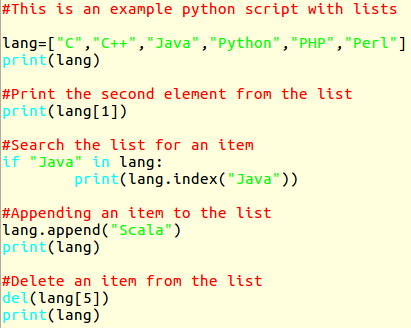


4.3.2. Results of 4.3.1 shown below;



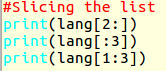
## 4.4. Deleting a list element

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



## 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;



## 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;

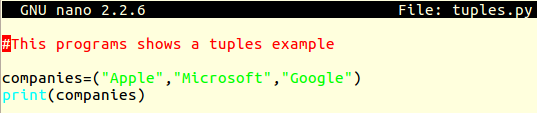


# 5. Tuples

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;



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;



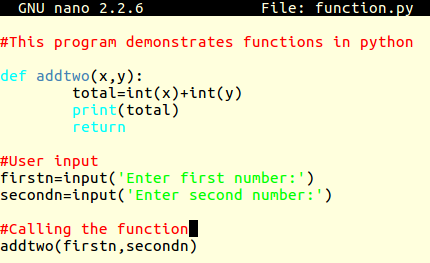
Several operations that are done in lists and tuples could also be applied for dictionaries.

# 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.



# 8. Reading a file

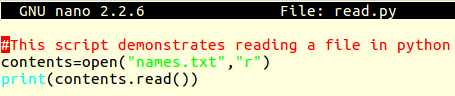
## 8.1. Create a file

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



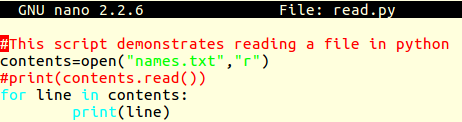
## 8.2. Read the entire file content

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



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

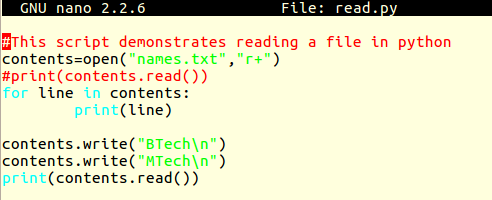
Modify the read.py script with the below code, to read the names.txt file line-by-line and print the contents;



# 9. Writing to a file

Modify the read.py script with the below code, to write contents to the names.txt file;

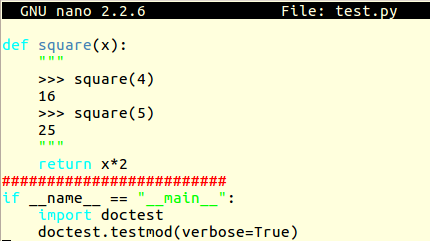
Check out the names.txt after running this read.py script.



# 10. Testing

## 10.1. doctest

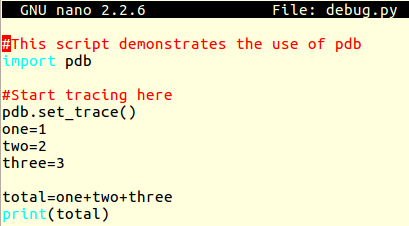
Write the following script (test.py), describing a function that squares the given number x. The function is tested using the doctest module of python. The docstring within the function defines the testing to be carried out.



Fix the script, so that both the tests pass.

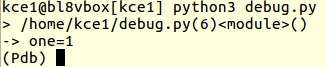
# 11. Debugging

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;



## 11.1. Tracing

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



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