Python Programming

Part 2

Stepping away from the interactive shell

We are now familiar with using the interactive shell to run Python instructions …

Time to start writing full Python programs using a text editor … e.g. notepad, Atom, sublime text

Interactive Shell	**Text Editor**
Useful for writing Python instructions, one at a time	Several instructions (lines of code) can be written and executed
Instructions are typed after the “>>>” prompt	No “>>>” prompt required

Our first Python program

1. Open the text editor and type
2. Type Print (‘Hello, world!’)
3. Save the file (make sure the file has the extension .py) – let’s save this as hello.py
4. Go back to the terminal and type: python hello.py to execute the program

… and so we have written and executed our first Python program

How long can a program be?

All lines of code in a program (file) will be executed. The program will only terminate (end or exit) when there are no more lines of code to be executed.

Our first program had only one line? Can we add a few more?

1. Ask the user their name
2. Print a message to say “Good afternoon” to the user
3. Tell the user how many characters are in their name.

Comments

Sometimes it is important to add notes or explanations to remind yourself/others of what the code is doing. These notes are called comments.

Adding a # in front of a line of code makes that line a comment.

Temporarily commenting out a line of code can also be used investigating why a program is not working as expected

Comments can be single line comments or multi-line comments

Exercise

1. Write a program to that accepts two numbers and finds the sum.

Flow Control

A program can execute several lines of instruction, one after another until it exhausts the lines. However, this is not the most fascinating thing about programming …

… a program can also make decisions on the following:

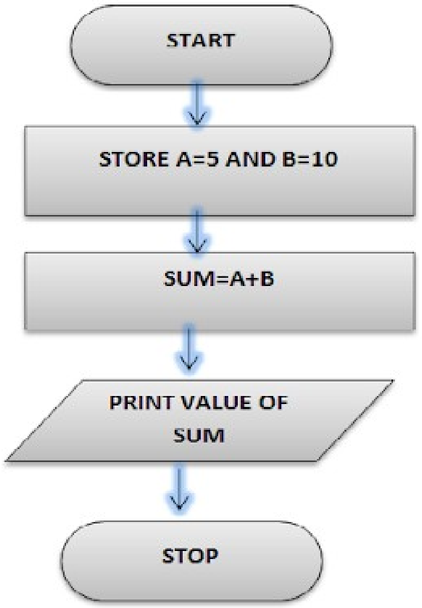
- Which lines should be executed

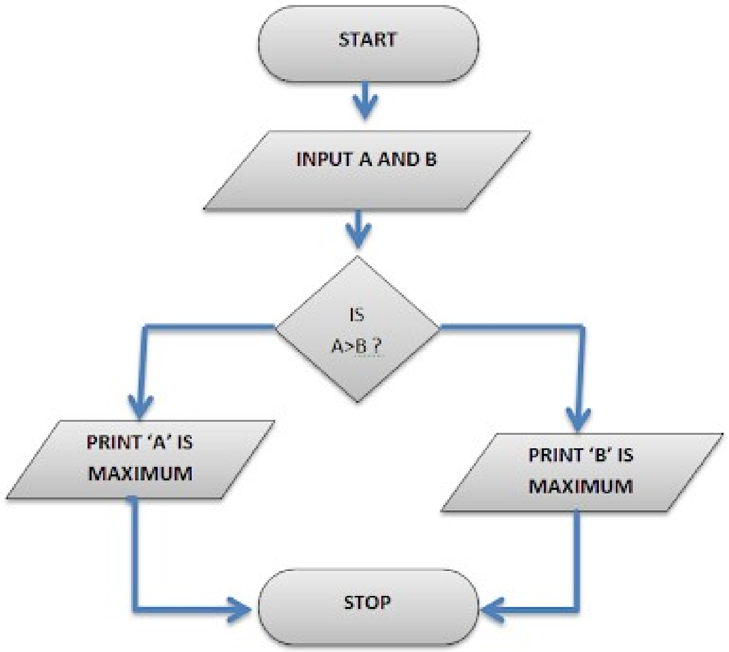
- Which lines to be skipped

- Should some lines be executed multiple times?

- At what point should the program terminate

Etc



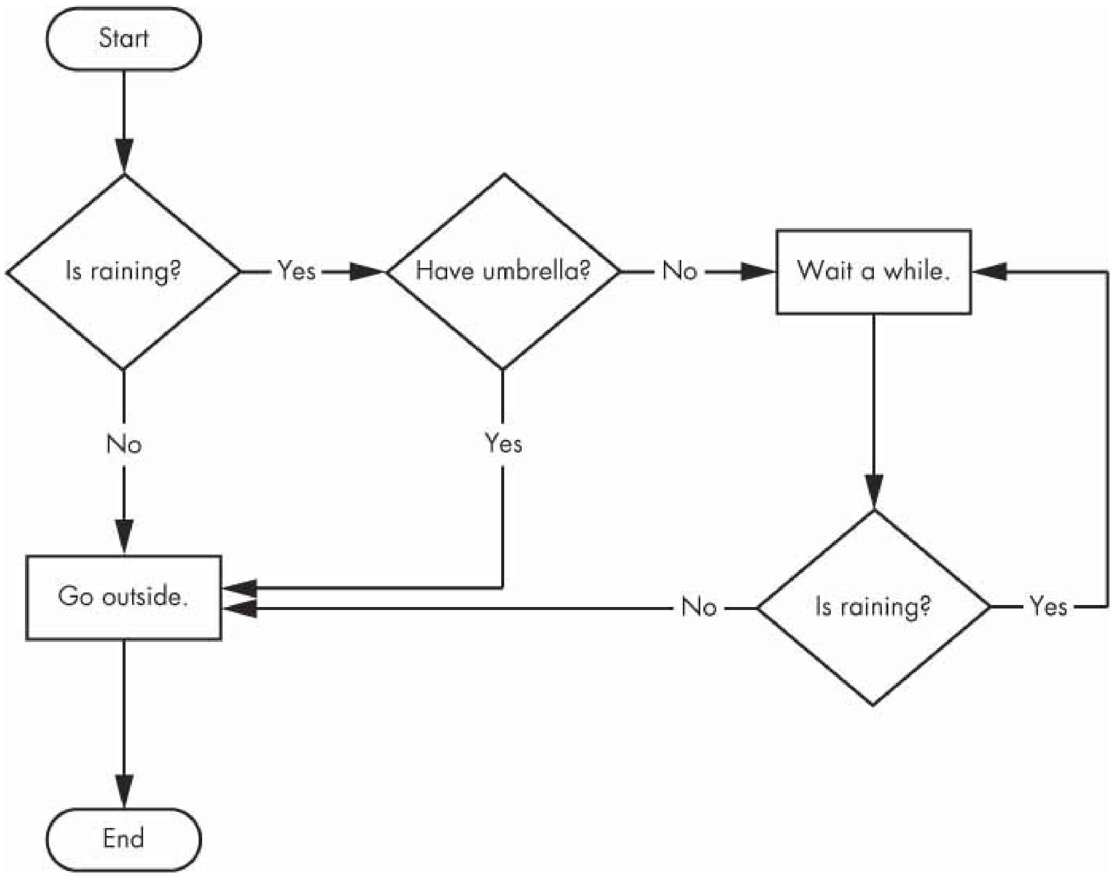


Source: http://clanguageprograming.blogspot.com/p/flowchart.html

Flow control statements

In practice, programs do not usually execute line-by-line from beginning to end … but how does the program make decisions?

Statements called “flow control” statements are used in deciding which instructions should be executed, and under what conditions



A flowchart to decide on what to do when it is raining. Source: https://automatetheboringstuff.com/2e/chapter2/

Boolean Values

We are already familiar with some Python data types (what are they?)

What is the number of possible values for each of these?

Another type of data type = Boolean data type → only two values "True" and "False"

Boolean values can be stored in variables just like any other data type (NOTE: but be mindful of the case, they are case sensitive)

e.g. x =True is not the same as x = true (try these in the interactive shell and see what you get)

PS: remember that True and False are <u>reserved words</u> in Python; and can't be used as variable names (try it!)

Comparison Operators

We’ve looked at arithmetic operators in Python (remember them?)... now let’s consider another group of operators…

Comparison operators (aka relational operators) are helpful when we need to compare two values and evaluate down to a single Boolean value e.g. is the value of x greater than 5?

Operator	Meaning
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

Playing with comparison operators

Depending on the values you use in a comparison operation, each comparison operator will evaluate to either True or False

Let’s try some out in the interactive shell:

```
64==64
64==62
‘hello’== ‘hello’
‘hello’ == ‘Hello’
‘dog’ == ‘cat’
‘dog’ != ‘cat’
```

```
True==True
True != False
True != False
45 == ‘45’
45 == 45.0
```

```
99 < 100
99>100
80 < 80
80<=80

Age=20
Age>=18
```

Note: = and == are not the same
Also, there is a subtle difference between the “==” operator and the “is” operator. “==” compares equality while “is” is an identity test to check if the operands refer to the same object or not (same memory location)

… and yet another kind of operator → Boolean operators

Just like comparison operators, Boolean operators evaluate to True or False.

Three Boolean operators:

1. **and**
2. **or**
3. **not**

“and” and “or” are classed as Binary Boolean operators - they take two Boolean values (or expressions)

The “not”, operator (unlike “and” and “or”), operates on only one Boolean value (or expression)

The Binary Boolean operators… “and” and “or”

“and” evaluates an expression to True, if both Boolean values are True, otherwise it evaluates to False … try it in the shell

True and True evaluates to ?
True and False evaluates to ?

The “or” operator evaluates an expression to True, if either of the two Boolean values is True (i.e at least one must be True), otherwise it evaluates to False … try it out

Truth tables

A <u>truth table</u> shows all possible results of a Boolean operator

Truth table for the and operator

Expression	Evaluates to ...
True and True	
True and False	
False and True	
False and False	

Truth table for the "or" operator

Expression	Evaluates to ...
True or True	
True or False	
False or True	
False or False	

The <u>not</u> operator

Do you remember the difference between the “not” operator and the Binary Boolean operators (“and” and “or”)? … we discussed this on slide 12

The “not” operator takes a Boolean value (or expression) and evaluates to its opposite.

The “not” operator is therefore a <u>unary</u> operator, as it operates on only one Boolean value (or expression)

```
not True
not False
not not True
not not not True
```

Combining Boolean Operators with Comparison Operators

Since both comparison operators and Boolean operators evaluate to Boolean values, we can combine both of them in expressions… let's try it.

Remember that the expressions will be evaluated from left to right; when the Boolean values for each expression have been found, the whole expression will be evaluated down to a single Boolean value

(3 < 5) and (8 < 10) will evaluate to ?

(5 < 6) and (7 < 4) will evaluate to ?

(1 ==5) or (4 ==4) will evaluate to ?

will evaluate to ?

Tip: Boolean operators use operator precedence too… arithmetic and comparison operators are evaluated first, then not operators, followed by and operators, and then or operators

Homework

1. What do the following expressions evaluate to

a. (8 > 3) and (5 == 10)

b. not (68 > 24)

c. (15 > 10) or (2 == 8)

d. not ((9 > 2) or (6 == 11))

e. (True and True) and (True == False)

f. (not False) or (not True)

g. (3 + 2 == 5) and (not (5 + 5 ==12)) and (2 *2 == 2 + 2)

2. Write a program that asks the user to enter 4 numbers. Your program should add the first 2 numbers and print out their sum as well as their product. Then your program should divide the 4th number by the 5th number and also display the result (add comments to your program and ensure that your outputs are easy to understand)

3. Complete the task in Assignment 4 of Part 1 as a single Python program. Try to add some comments to your code to make it easier to understand.