1. Getting Started

1.1. Running Python Interpreter

Python comes with an interactive interpreter. When you type python in your shell or command prompt, the python interpreter becomes active with a >>> prompt and waits for your commands.

$ python

Python 2.7.1 (r271:86832, Mar 17 2011, 07:02:35)

[GCC 4.2.1 (Apple Inc. build 5664)] on darwin

Type "help", "copyright", "credits" or "license" for more information.

>>>

Now you can type any valid python expression at the prompt. python reads the typed expression, evaluates it and prints the result.

>>> 42

42

>>> 4 + 2

6

**Problem 1:**Open a new Python interpreter and use it to find the value of 2 + 3.

1.2. Running Python Scripts

Open your text editor, type the following text and save it as hello.py.

**print** "hello, world!"

And run this program by calling python hello.py. Make sure you change to the directory where you saved the file before doing it.

anand@bodhi ~$ python hello.py

hello, world!

anand@bodhi ~$

Text after # character in any line is considered as comment.

*# This is helloworld program*

*# run this as:*

*# python hello.py*

**print** "hello, world!"

**Problem 2:**Create a python script to print hello, world! four times.

**Problem 3:**Create a python script with the following text and see the output.

1 + 2

If it doesn’t print anything, what changes can you make to the program to print the value?

Ads not by this site

1.3. Assignments

One of the building blocks of programming is associating a name to a value. This is called assignment. The associated name is usually called a *variable*.

>>> x = 4

>>> x \* x

16

In this example x is a variable and it’s value is 4.

If you try to use a name that is not associated with any value, python gives an error message.

>>> foo

**Traceback (most recent call last):**

File "<stdin>", line 1, in ?

NameError: name 'foo' is not defined

>>> foo = 4

>>> foo

4

If you re-assign a different value to an existing variable, the new value overwrites the old value.

>>> x = 4

>>> x

4

>>> x = 'hello'

>>> x

'hello'

It is possible to do multiple assignments at once.

>>> a**,** b = 1**,** 2

>>> a

1

>>> b

2

>>> a + b

3

Swapping values of 2 variables in python is very simple.

>>> a**,** b = 1**,** 2

>>> a**,** b = b**,** a

>>> a

2

>>> b

1

When executing assignments, python evaluates the right hand side first and then assigns those values to the variables specified in the left hand side.

**Problem 4:**What will be output of the following program.

x = 4

y = x + 1

x = 2

**print** x**,** y

**Problem 5:**What will be the output of the following program.

x**,** y = 2**,** 6

x**,** y = y**,** x + 2

**print** x**,** y

**Problem 6:**What will be the output of the following program.

a**,** b = 2**,** 3

c**,** b = a**,** c + 1

**print** a**,** b**,** c

1.4. Numbers

We already know how to work with numbers.

>>> 42

42

>>> 4 + 2

6

Python also supports decimal numbers.

>>> 4.2

4.2

>>> 4.2 + 2.3

6.5

Python supports the following operators on numbers.

* + addition
* - subtraction
* \* multiplication
* / division
* \*\* exponent
* % remainder

Let’s try them on integers.

>>> 7 + 2

9

>>> 7 - 2

5

>>> 7 \* 2

14

>>> 7 / 2

3

>>> 7 \*\* 2

49

>>> 7 % 2

1

If you notice, the result 7 / 2 is 3 not 3.5. It is because the / operator when working on integers, produces only an integer. Lets see what happens when we try it with decimal numbers:

>>> 7.0 / 2.0

3.5

>>> 7.0 / 2

3.5

>>> 7 / 2.0

3.5

The operators can be combined.

>>> 7 + 2 + 5 - 3

11

>>> 2 \* 3 + 4

10

It is important to understand how these compound expressions are evaluated. The operators have precedence, a kind of priority that determines which operator is applied first. Among the numerical operators, the precedence of operators is as follows, from low precedence to high.

* +, -
* \*, /, %
* \*\*

When we compute 2 + 3 \* 4, 3 \* 4 is computed first as the precedence of \* is higher than + and then the result is added to 2.

>>> 2 + 3 \* 4

14

We can use parenthesis to specify the explicit groups.

>>> **(**2 + 3**)** \* 4

20

All the operators except \*\* are left-associcate, that means that the application of the operators starts from left to right.

1 + 2 + 3 \* 4 + 5

↓

3 + 3 \* 4 + 5

↓

3 + 12 + 5

↓

15 + 5

↓

20

1.5. Strings

Strings what you use to represent text.

Strings are a sequence of characters, enclosed in single quotes or double quotes.

>>> x = "hello"

>>> y = 'world'

>>> **print** x**,** y

hello world

There is difference between single quotes and double quotes, they can used interchangebly.

Multi-line strings can be written using three single quotes or three double quotes.

x = """This is a multi-line string

written in

three lines."""

**print** x

y = '''multi-line strings can be written

using three single quote characters as well.

The string can contain 'single quotes' or "double quotes"

in side it.'''

**print** y

Ads not by this site

1.6. Functions

Just like a value can be associated with a name, a piece of logic can also be associated with a name by defining a function.

>>> **def** square**(**x**):**

... **return** x \* x

...

>>> square**(**5**)**

25

The body of the function is indented. Indentation is the Python’s way of grouping statements.

The ... is the secondary prompt, which the Python interpreter uses to denote that it is expecting some more input.

The functions can be used in any expressions.

>>> square**(**2**)** + square**(**3**)**

13

>>> square**(**square**(**3**))**

81

Existing functions can be used in creating new functions.

>>> **def** sum\_of\_squares**(**x**,** y**):**

... **return** square**(**x**)** + square**(**y**)**

...

>>> sum\_of\_squares**(**2**,** 3**)**

13

Functions are just like other values, they can assigned, passed as arguments to other functions etc.

>>> f = square

>>> f**(**4**)**

16

>>> **def** fxy**(**f**,** x**,** y**):**

... **return** f**(**x**)** + f**(**y**)**

...

>>> fxy**(**square**,** 2**,** 3**)**

13

It is important to understand, the scope of the variables used in functions.

Lets look at an example.

x = 0

y = 0

**def** incr**(**x**):**

y = x + 1

**return** y

incr**(**5**)**

**print** x**,** y

Variables assigned in a function, including the arguments are called the local variables to the function. The variables defined in the top-level are called global variables.

Changing the values of x and y inside the function incr won’t effect the values of globalx and y.

But, we can use the values of the global variables.

pi = 3.14

**def** area**(**r**):**

**return** pi \* r \* r

When Python sees use of a variable not defined locally, it tries to find a global variable with that name.

However, you have to explicitly declare a variable as global to modify it.

numcalls = 0

**def** square**(**x**):**

**global** numcalls

numcalls = numcalls + 1

**return** x \* x

**Problem 7:**How many multiplications are performed when each of the following lines of code is executed?

**print** square**(**5**)**

**print** square**(**2\*5**)**

**Problem 8:**What will be the output of the following program?

x = 1

**def** f**():**

**return** x

**print** x

**print** f**()**

**Problem 9:**What will be the output of the following program?

x = 1

**def** f**():**

x = 2

**return** x

**print** x

**print** f**()**

**print** x

**Problem 10:**What will be the output of the following program?

x = 1

**def** f**():**

y = x

x = 2

**return** x + y

**print** x

**print** f**()**

**print** x

**Problem 11:**What will be the output of the following program?

x = 2

**def** f**(**a**):**

x = a \* a

**return** x

y = f**(**3**)**

**print** x**,** y

Functions can be called with keyword arguments.

>>> **def** difference**(**x**,** y**):**

... **return** x - y

...

>>> difference**(**5**,** 2**)**

3

>>> difference**(**x=5**,** y=2**)**

3

>>> difference**(**5**,** y=2**)**

3

>>> difference**(**y=2**,** x=5**)**

3

And some arguments can have default values.

>>> **def** increment**(**x**,** amount=1**):**

... **return** x + amount

...

>>> **def** increment**(**10**)**

11

>>> increment**(**10**,** 5**)**

15

>>> increment**(**10**,** amount=2**)**

12

There is another way of creating functions, using the lambda operator.

>>> cube = **lambda** x**:** x \*\* 3

>>> fxy**(**cube**,** 2**,** 3**)**

35

>>> fxy**(lambda** x**:** x \*\* 3**,** 2**,** 3**)**

35

Notice that unlike function defination, lambda doesn’t need a return. The body of thelambda is a single expression.

The lambda operator becomes handy when writing small functions to be passed as arguments etc. We’ll see more of it as we get into solving more serious problems.

1.6.1. Built-in Functions

Python provides some useful built-in functions.

>>> min**(**2**,** 3**)**

2

>>> max**(**3**,** 4**)**

4

The built-in function len computes length of a string.

>>> len**(**"helloworld"**)**

10

The built-in function int converts string to ingeter and built-in function str converts integers and other type of objects to strings.

>>> int**(**"50"**)**

50

>>> str**(**123**)**

"123"

**Problem 12:**Write a function count\_digits to find number of digits in the given number.

>>> count\_digits**(**5**)**

1

>>> count\_digits**(**12345**)**

5

1.6.2. Methods

Methods are special kind of functions that work on an object.

For example, upper is a method available on string objects.

>>> x = "hello"

>>> **print** x.upper**()**

HELLO

As already mentioned, methods are also functions. They can be assigned to other variables can be called separately.

>>> f = x.upper

>>> **print** f**()**

HELLO

**Problem 13:**Write a function *istrcmp* to compare two strings, ignoring the case.

>>> istrcmp**(**'python'**,** 'Python'**)**

True

>>> istrcmp**(**'LaTeX'**,** 'Latex'**)**

True

>>> istrcmp**(**'a'**,** 'b'**)**

False

1.7. Conditional Expressions

Python provides various operators for comparing values. The result of a comparison is a boolean value, either True or False.

>>> 2 < 3

False

>>> 2 > 3

True

Here is the list of available conditional operators.

* == equal to
* != not equal to
* < less than
* > greater than
* <= less than or equal to
* >= greater than or equal to

It is even possible to combine these operators.

>>> x = 5

>>> 2 < x < 10

True

>>> 2 < 3 < 4 < 5 < 6

True

The conditional operators work even on strings - the ordering being the lexical order.

>>> "python" > "perl"

True

>>> "python" > "java"

True

There are few logical operators to combine boolean values.

* a and b is True only if both a and b are True.
* a or b is True if either a or b is True.
* not a is True only if a is False.

>>> True **and** True

True

>>> True **and** False

False

>>> 2 < 3 **and** 5 < 4

False

>>> 2 < 3 **or** 5 < 4

True

**Problem 14:**What will be output of the following program?

**print** 2 < 3 **and** 3 > 1

**print** 2 < 3 **or** 3 > 1

**print** 2 < 3 **or** **not** 3 > 1

**print** 2 < 3 **and** **not** 3 > 1

**Problem 15:**What will be output of the following program?

x = 4

y = 5

p = x < y **or** x < z

**print** p

**Problem 16:**What will be output of the following program?

True**,** False = False**,** True

**print** True**,** False

**print** 2 < 3

1.7.1. The if statement

The if statement is used to execute a piece of code only when a boolean expression is true.

>>> x = 42

>>> **if** x % 2 == 0**:** **print** 'even'

even

>>>

In this example, print 'even' is executed only when x % 2 == 0 is True.

The code associated with if can be written as a separate indented block of code, which is often the case when there is more than one statement to be executed.

>>> **if** x % 2 == 0**:**

... **print** 'even'

...

even

>>>

The if statement can have optional else clause, which is executed when the boolean expression is False.

>>> x = 3

>>> **if** x % 2 == 0**:**

... **print** 'even'

... **else:**

... **print** 'odd'

...

odd

>>>

The if statement can have optional elif clauses when there are more conditions to be checked. The elif keyword is short for else if, and is useful to avoid excessive indentation.

>>> x = 42

>>> **if** x < 10**:**

... **print** 'one digit number'

... **elif** x < 100**:**

... **print** 'two digit number'

... **else:**

... **print** 'big number'

...

two digit number

>>>

**Problem 17:**What happens when the following code is executed? Will it give any error? Explain the reasons.

x = 2

**if** x == 2**:**

**print** x

**else:**

**print** y

**Problem 18:**What happens the following code is executed? Will it give any error? Explain the reasons.

x = 2

**if** x == 2**:**

**print** x

**else:**

x +

1.8. Lists

Lists are one of the great datastructures in Python. We are going to learn a little bit about lists now. Basic knowledge of lists is requrired to be able to solve some problems that we want to solve in this chapter.

Here is a list of numbers.

>>> x = **[**1**,** 2**,** 3**]**

And here is a list of strings.

>>> x = **[**"hello"**,** "world"**]**

List can be heterogeneous. Here is a list containings integers, strings and another list.

>>> x = **[**1**,** 2**,** "hello, "world", ["another", "list"]]

The built-in function len works for lists as well.

>>> x = **[**1**,** 2**,** 3**]**

>>> len**(**x**)**

3

The [] operator is used to access individual elements of a list.

>>> x = **[**1**,** 2**,** 3**]**

>>> x**[**1**]**

2

>>> x**[**1**]** = 4

>>> x**[**1**]**

4

The first element is indexed with 0, second with 1 and so on.

We’ll learn more about lists in the next chapter.

Ads not by this site

1.9. Modules

Modules are libraries in Python. Python ships with many standard library modules.

A module can be imported using the import statement.

Lets look at time module for example:

>>> **import** time

>>> time.asctime**()**

'Tue Sep 11 21:42:06 2012'

The asctime function from the time module returns the current time of the system as a string.

The sys module provides access to the list of arguments passed to the program, among the other things.

The sys.argv variable contains the list of arguments passed to the program. As a convention, the first element of that list is the name of the program.

Lets look at the following program echo.py that prints the first argument passed to it.

**import** sys

**print** sys.argv**[**1**]**

Lets try running it.

$ python echo.py hello

hello

$ python echo.py hello world

hello

There are many more interesting modules in the standard library. We’ll learn more about them in the coming chapters.

**Problem 19:**Write a program add.py that takes 2 numbers as command line arguments and prints its sum.

$ python add.py 3 5

8

$ python add.py 2 9

11

About this Book

This book is prepared from the training notes of [Anand Chitipothu](http://anandology.com/).

Anand conducts Python trainings classes on a semi-regular basis in Bangalore, India. Checkout out the[upcoming trainings](http://anandology.com/trainings/) if you are interested.

[Table Of Contents](http://anandology.com/python-practice-book/index.html)

* [1. Getting Started](http://anandology.com/python-practice-book/getting-started.html)
  + [1.1. Running Python Interpreter](http://anandology.com/python-practice-book/getting-started.html#running-python-interpreter)
  + [1.2. Running Python Scripts](http://anandology.com/python-practice-book/getting-started.html#running-python-scripts)
  + [1.3. Assignments](http://anandology.com/python-practice-book/getting-started.html#assignments)
  + [1.4. Numbers](http://anandology.com/python-practice-book/getting-started.html#numbers)
  + [1.5. Strings](http://anandology.com/python-practice-book/getting-started.html#strings)
  + [1.6. Functions](http://anandology.com/python-practice-book/getting-started.html#functions)
    - [1.6.1. Built-in Functions](http://anandology.com/python-practice-book/getting-started.html#built-in-functions)
    - [1.6.2. Methods](http://anandology.com/python-practice-book/getting-started.html#methods)
  + [1.7. Conditional Expressions](http://anandology.com/python-practice-book/getting-started.html#conditional-expressions)
    - [1.7.1. The if statement](http://anandology.com/python-practice-book/getting-started.html#the-if-statement)
  + [1.8. Lists](http://anandology.com/python-practice-book/getting-started.html#lists)
  + [1.9. Modules](http://anandology.com/python-practice-book/getting-started.html#modules)

Related Topics

* [Documentation overview](http://anandology.com/python-practice-book/index.html)
  + Previous: [Python Practice Book](http://anandology.com/python-practice-book/index.html)
  + Next: [2. Working with Data](http://anandology.com/python-practice-book/working-with-data.html)

This Page

* [Show Source](http://anandology.com/python-practice-book/_sources/getting-started.txt)

Quick search

Top of Form



Bottom of Form

Enter search terms or a module, class or function name.

© Copyright 2012, [Anand Chitipothu](http://anandology.com/). Created using [Sphinx](http://sphinx.pocoo.org/).

Ads not by this site

Ads not by this site

Ads not by this site