Introduction to Programming Using Python

Microsoft Python Certification | **Exam 98-381**

Lecture 3:

Calculations

Lecturer: **James W. Jiang**, Ph.D. | Summer, 2018

Microsoft™

SAVVY PRO
PROFESSIONAL EDUCATION TRAINING

Basic Operations:
Arithmetic Operators

most common math operations

Symbol	Operation	Example
+	Addition	5+2 = 7
-	Subtraction	5-2 = 3
*	Multiplication	5*2 = 10
/	Division	5/2 = 2.5
**	Exponent	5**2 = 25
%	Modulo	5%2 = 1

PROFESSIONAL EDUCATION TRAINING

most common math operations

```
40 + 2 # addition
42

43 - 1 # subtraction
42

6 * 7  # multiplication
42

84 / 2  # division
42.0

6 ** 2 + 6 # operator ** performs exponentiation
42
```


PROFESSIONAL EDUCATION TRAINING

most common math operations

```
width = 20
height = 5

area = width * height

perimeter = 2*width + 2*height

perimeter = 2*(width+height)
```


PROFESSIONAL EDUCATION TRAINING

Floor Division

The floor division operator, //, divides two numbers and rounds down to an integer. But we don't normally write hours with decimal points. Floor division returns the integer number of hours, dropping the fraction part

```
minutes = 105

hours = minutes // 60

hours

1
```


PROFESSIONAL EDUCATION TRAINING

modulus operator

An alternative is to use the modulus operator, %, which divides two numbers and returns the remainder:

```
minutes = 105

remainder = minutes % 60

remainder

45
```

Exponentiation

In some other languages, ^ is used for exponentiation, but in Python it is a bitwise operator called XOR. If you are not familiar with bitwise operators, the result will surprise you:

```
6 ^ 2
```

```
4
```


PROFESSIONAL EDUCATION TRAINING

Basic Operations:
Relational Operators

relational operators

A boolean expression is an expression that is either true or false

```
x == y   # x is equal to y
x != y   # x is not equal to y
x >  y   # x is greater than y
x <  y   # x is less than y
x >= y   # x is greater than or equal to y
x <= y   # x is less than or equal to y
x <> y   # This is similar to != operator.
```


PROFESSIONAL EDUCATION TRAINING

Basic Operations:
Logical operators

Logical Operators

There are three logical operators: **and**, **or**, and **not**.

```
x > 0 and x < 10
# is true only if x is greater than 0 and less than 10


n%2 == 0 or n%3 == 0
# is true if either or both of the conditions is true,
# that is, if the number is divisible by 2 or 3
```


PROFESSIONAL EDUCATION TRAINING

Logical Operators

Operator	Meaning	Example
`and`	True if both the operands are true	`x and y`
`or`	True if either of the operands is true	`x or y`
`not`	True if operand is false (complements the operand)	`not x`

PROFESSIONAL EDUCATION TRAINING

Logical Operators

```
x = True
y = False
# Output: x and y is False
print('x and y is',x and y)

# Output: x or y is True
print('x or y is',x or y)

# Output: not x is False
print('not x is',not x)
```

```
In [46]: x = True
    ...: y = False
    ...: # Output: x and y is False
    ...: print('x and y is',x and y)
    ...:
    ...: # Output: x or y is True
    ...: print('x or y is',x or y)
    ...:
    ...: # Output: not x is False
    ...: print('not x is',not x)
    ...:
x and y is False
x or y is True
not x is False
```


PROFESSIONAL EDUCATION TRAINING

Logical Operators

```
>>> not False
True
>>> not True
False
>>> True and True
True
>>> True and False
False
>>> True or False
True
```

Logical Operators

```
>>> False or False

False

>>> not(False or False)

True

>>> not(False and False)

True

>>> not(False and True)

True
```


PROFESSIONAL EDUCATION TRAINING

Basic Operations:
Formatting Numbers

PROFESSIONAL EDUCATION TRAINING

Formatting Numbers

Sometimes you will need to format the numbers when you display them to users

Syntax	Output
`print('I have %d cats' % 6)`	`I have 6 cats`
`print('I have %3d cats' % 6)`	`I have   6 cats`
`print('I have %03d cats' % 6)`	`I have 006 cats`
`print('I have %f cats' % 6)`	`I have 6.000000 cats`
`print('I have %.2f cats' % 6)`	`I have 6.00 cats`

PROFESSIONAL EDUCATION TRAINING

Formatting Numbers

You can also use a format method to format numeric values

Syntax	Output
`print("I have {0:d} cats".format(6))`	`I have 6 cats`
`print("I have {0:3d} cats".format(6))`	`I have   6 cats`
`print("I have {0:03d} cats".format(6))`	`I have 006 cats`
`print("I have {0:f} cats".format(6))`	`I have 6.000000 cats`
`print("I have {0:.2f} cats".format(6))`	`I have 6.00 cats`

PROFESSIONAL EDUCATION TRAINING

Basic Operations:
Conversion

PROFESSIONAL EDUCATION TRAINING

Conversion

What is wrong with the script:

```
salary = input("Please enter your salary: ")
bonus = input("Please enter your bonus: ")
payCheck = salary + bonus
print(payCheck)
```

```
In [18]: salary = input("Please enter your salary: ")
    ...: bonus = input("Please enter your bonus: ")
    ...: payCheck = salary + bonus
    ...: print(payCheck)
    ...:

Please enter your salary: 100

Please enter your bonus: 100
100100
```


PROFESSIONAL EDUCATION TRAINING

Conversion

```
salary = 100
bonus = 100
payCheck = salary + bonus
print(payCheck)
```

```
In [19]: salary = 100
    ...: bonus = 100
    ...: payCheck = salary + bonus
    ...: print(payCheck)
    ...:
200
```


PROFESSIONAL EDUCATION TRAINING

Types of Conversions

We need a way to tell our program we want to treat values as a number instead of a string

There are functions to convert from one datatype to another.

```
int(value)          # converts to an integer
long(value)         # converts to a long integer
float(value)        # converts to a floating number (i.e.
a number that can hold decimal places)
str(value)          # converts to a string
```


PROFESSIONAL EDUCATION TRAINING

Conversion

```
salary = input("Please enter your salary: ")
bonus = input("Please enter your bonus: ")
payCheck = salary + bonus
payCheck = float(salary) + float(bonus)
print(payCheck)
```

```
In [20]: salary = input("Please enter your salary: ")
    ...: bonus = input("Please enter your bonus: ")
    ...: payCheck = salary + bonus
    ...: payCheck = float(salary) + float(bonus)
    ...: print(payCheck)
    ...:

Please enter your salary: 100

Please enter your bonus: 100
200.0

In [21]:
```


PROFESSIONAL EDUCATION TRAINING

Basic Operations:
Assignment Operators

Assignment Operations

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+=	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-=	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*=	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a

PROFESSIONAL EDUCATION TRAINING

Assignment Operations

Operator	Description	Example
/=	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / a
%=	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**=	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//=	It performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

PROFESSIONAL EDUCATION TRAINING

Assignment Operations

```
a = 21; b = 10; c = 0; c = a + b

print ("Line 1 - Value of c is ", c)

c += a

print ("Line 2 - Value of c is ", c)

c *= a

print ("Line 3 - Value of c is ", c)
```

```
In [49]: a = 21
    ...: b = 10
    ...: c = 0
    ...: c = a + b
    ...: print ("Line 1 - Value of c is ", c)
    ...: c += a
    ...: print ("Line 2 - Value of c is ", c)
    ...: c *= a
    ...: print ("Line 3 - Value of c is ", c)
    ...:
Line 1 - Value of c is  31
Line 2 - Value of c is  52
Line 3 - Value of c is  1092
```


PROFESSIONAL EDUCATION TRAINING

Assignment Operations

```
c /= a
print ("Line 4 - Value of c is ", c)
c = 2
c %= a
print ("Line 5 - Value of c is ", c)
c **= a
print ("Line 6 - Value
c //= a
print ("Line 7 - Value
```

```
In [50]: c /= a
    ...: print ("Line 4 - Value of c is ", c)
    ...: c = 2
    ...: c %= a
    ...: print ("Line 5 - Value of c is ", c)
    ...: c **= a
    ...: print ("Line 6 - Value of c is ", c)
    ...: c //= a
    ...: print ("Line 7 - Value of c is ", c)
    ...:
Line 4 - Value of c is  52.0
Line 5 - Value of c is  2
Line 6 - Value of c is  2097152
Line 7 - Value of c is  99864
```

Basic Operations:
Mathematical Functions

PROFESSIONAL EDUCATION TRAINING

Built-in Mathematical Functions

Function	Description
`abs(x)`	The absolute value of x: the (positive) distance between x and zero.
`ceil(x)`	The ceiling of x: the smallest integer not less than x
`cmp(x,y)`	-1 if x < y, 0 if x == y, or 1 if x > y
`exp(x)`	The exponential of x: e^x
`fabs(x)`	The absolute value of x.
`floor(x)`	The floor of x: the largest integer not greater than x
`log(x)`	The natural logarithm of x, for x> 0

PROFESSIONAL EDUCATION TRAINING

Built-in Mathematical Functions

Function	Description
`log10(x)`	The base-10 logarithm of x for x> 0.
`max(x1, x2,...)`	The largest of its arguments: the value closest to positive infinity
`min(x1, x2,...)`	The smallest of its arguments: the value closest to negative infinity
`modf(x)`	The fractional and integer parts of x in a two-item tuple. Both parts have the same sign as x. The integer part is returned as a float.
`pow(x, y)`	The value of x**y.
`round(x [,n])`	x rounded to n digits from the decimal point. Python rounds away from zero as a tie-breaker: round(0.5) is 1.0 and round(-0.5) is -1.0.
`sqrt(x)`	The square root of x for x > 0

Basic Operations:

Bitwise Operators

PROFESSIONAL EDUCATION TRAINING

Bitwise Operators

Bitwise operator works on bits and performs bit by bit operation. Assume if a = 60; and b = 13; Now in binary format they will be as follows

```
a = 0011 1100

b = 0000 1101
```

Operator	Description	Example
& Binary AND	Operator copies a bit to the result if it exists in both operands	(a & b) (means 0000 1100)
| Binary OR	It copies a bit if it exists in either operand.	(a | b) = 61 (means 0011 1101)

PROFESSIONAL EDUCATION TRAINING

Bitwise Operators

Operator	Description	Example
^ Binary XOR	It copies the bit if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.	(~a) = -61 (means 1100 0011 in 2's complement form due to a signed binary number.
<< Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.	a << 2 = 240 (means 1111 0000)
>> Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.	a >> 2 = 15 (means 0000 1111)

PROFESSIONAL EDUCATION TRAINING

Basic Operations:
Membership Operators

in and not in

Python's membership operators test for membership in a sequence, such as strings, lists, or tuples. There are two membership operators as explained below

Operator	Description	Example
`in`	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
`not in`	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

PROFESSIONAL EDUCATION TRAINING

in

```
a = 10
b = 20
list = [1, 2, 3, 4, 5 ];
if ( a in list ):
   print "Line 1 - a is available in the given list"
else:
   print "Line 1 - a is not available in the given list"
```


PROFESSIONAL EDUCATION TRAINING

not in

```
if ( b not in list ):
   print "Line 2 - b is not available in the given list"
else:
   print "Line 2 - b is available in the given list"
a = 2
if ( a in list ):
   print "Line 3 - a is available in the given list"
else:
   print "Line 3 - a is not available in the given list"
```

Basic Operations:
Identity Operators

is and is not

Identity operators compare the memory locations of two objects. There are two Identity operators explained below

Operator	Description	Example
`is`	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here **is** results in 1 if id(x) equals id(y).
`is not`	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here **is not** results in 1 if id(x) is not equal to id(y).

PROFESSIONAL EDUCATION TRAINING

is and is not

```
x1 = 5; y1 = 5
x2 = 'Hello'; y2 = 'Hello'
x3 = [1,2,3]
y3 = [1,2,3]
# Output: False
print(x1 is not y1)
# Output: True
print(x2 is y2)
# Output: False
print(x3 is y3)
```

PROFESSIONAL EDUCATION TRAINING

Basic Operations:

Orders of Operations

Order of operations

In the United States, the acronym PEMDAS is common. It stands for Parentheses, Exponents, Multiplication/Division, Addition/Subtraction. PEMDAS is often expanded to the mnemonic "Please Excuse My Dear Aunt Sally".

Level	Category	Operators
7(high)	exponent	`**`
6	multiplication	`*,/,//,%`
5	addition	`+,-`
4	relational	`==,!=,<=,>=,>,<`
3	logical	`not`
2	logical	`and`
1(low)	logical	`or`

PROFESSIONAL EDUCATION TRAINING

Order of operations

Operator	Description
()	Parentheses (grouping)
f(args...)	Function call
x[index:index]	Slicing
x[index]	Subscription
x.attribute	Attribute reference
**	Exponentiation
~x	Bitwise not
+x, -x	Positive, negative
*, /, %	Multiplication, division, remainder
+, -	Addition, subtraction
<<, >>	Bitwise shifts

PROFESSIONAL EDUCATION TRAINING

Order of operations, cont'd

Operator	Description
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
in, not in, is, is not, <, <=, >, >=, <>, !=, ==	Comparisons, membership, identity
not x	Boolean NOT
and	Boolean AND
or	Boolean OR
lambda	Lambda expression

PROFESSIONAL EDUCATION TRAINING

Order of operations

```
True is 1

True is 1.0

True == 1

1 == True

True == 2

False is 0

False == 0

False == 0.0

True + 1

int(True)

int(False)
```


PROFESSIONAL EDUCATION TRAINING

Order of operations

```
'OK' or 'KO'

'OK' and 'KO'

True and 'OK' or 'KO'

True and 'OK' and 'KO'

True or 'OK' or 'KO'

False and 'OK' or 'KO'

False and 'OK' and 'KO'

False or 'OK' or 'KO'
```


PROFESSIONAL EDUCATION TRAINING

Order of operations

```
0 and 0 or 1

0 and (0 or 1)

1 or 0 and 0

(1 or 0) and 0

0 and False or 1

0 and False or True

False and False or True
```


PROFESSIONAL EDUCATION TRAINING

Modules:
Modules

Modules

Modules refer to a file containing Python statements and definitions. A file containing Python code, for e.g.: example.py, is called a module and its module name would be example. We use modules to break down large programs into small manageable and organized files. Furthermore, modules provide reusability of code. We can define our most used functions in a module and import it, instead of copying their definitions into different programs.

Create a Module

Let us create a module. Type the following and save it as example.py.

```
# Python Module example
# saved in C:\Users\James\Anaconda3\lib
def add(a, b):
    """This program adds two
    numbers and return the result"""
    result = a + b
    return result
```

Here, we have defined a function add() inside a module named example. The function takes in two numbers and returns their sum.

PROFESSIONAL EDUCATION TRAINING

import modules in Python

We can import the definitions inside a module to another module or the interactive interpreter in Python.

We use the import keyword to do this. To import our previously defined module example we type the following in the Python prompt.

```
import example
```

This does not enter the names of the functions defined in example directly in the current symbol table. It only enters the module name example there.

Using the module name we can access the function using dot (.) operation. For example:

```
example.add(4,5.5)
```

PROFESSIONAL EDUCATION TRAINING

Python Standard Modules

You can check out the full list of Python standard modules and what they are for. These files are in the Lib directory inside the location where you installed Python.

`https://docs.python.org/3/py-modindex.html#cap-p`

Python Module Index

_ | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | z

_	
__future__	*Future statement definitions*
__main__	*The environment where the top-level script is run.*
_dummy_thread	*Drop-in replacement for the _thread module.*
_thread	*Low-level threading API.*
a	
abc	*Abstract base classes according to PEP 3119.*
aifc	*Read and write audio files in AIFF or AIFC format.*
argparse	*Command-line option and argument parsing library.*
array	*Space efficient arrays of uniformly typed numeric values.*
ast	*Abstract Syntax Tree classes and manipulation.*
asynchat	*Support for asynchronous command/response protocols.*

PROFESSIONAL EDUCATION TRAINING

Modules:

Python import statement

Import Existing Modules

We can import a module using import statement and access the definitions inside it using the dot operator as described above. Here is an example.

```
# import statement example

# to import standard module math

import math

print("The value of pi is", math.pi)
```


PROFESSIONAL EDUCATION TRAINING

Import with renaming

We can import a module by renaming it as follows.

```
# import module by renaming it
import math as m
print("The value of pi is", m.pi)
```

We have renamed the math module as m. This can save us typing time in some cases. In this case, note that the name math is not recognized in our scope. Hence, math.pi is invalid, m.pi is the correct implementation.

PROFESSIONAL EDUCATION TRAINING

import specific names from a module

We can import specific names from a module without importing the module as a whole. Here is an example.

```
# import only pi from math module

from math import pi

print("The value of pi is", pi)
```

We imported only the attribute pi from the module. In such case we don't use the dot operator. We could have imported multiple attributes as follows.

```
from math import pi, e

pi

e
```


PROFESSIONAL EDUCATION TRAINING

Import all names

We can import all names(definitions) from a module using the following construct.

```
# import all names from the standard module math
from math import *
print("The value of pi is", pi)
```

We imported all the definitions from the math module. This makes all names except those beginning with an underscore, visible in our scope.

Importing everything with the asterisk (*) symbol is not a good programming practice. This can lead to duplicate definitions for an identifier. It also hampers the readability of our code.

Modules:

Python Module Search Path

Search Paths

While importing a module, Python looks at several places. Interpreter first looks for a built-in module then (if not found) into a list of directories defined in sys.path. The search is in this order:

- The current directory.
- PYTHONPATH (an environment variable with a list of directory).
- The installation-dependent default directory.

PROFESSIONAL EDUCATION TRAINING

Search Paths

```
import sys

sys.path
```

```
In [30]: sys.path
Out[30]:
['',
 'C:\\Users\\James\\Anaconda3\\lib\\site-packages\\spyder\\utils\\site',
 'C:\\Users\\James\\Anaconda3\\python36.zip',
 'C:\\Users\\James\\Anaconda3\\DLLs',
 'C:\\Users\\James\\Anaconda3\\lib',
 'C:\\Users\\James\\Anaconda3',
 'C:\\Users\\James\\Anaconda3\\lib\\site-packages',
 'C:\\Users\\James\\Anaconda3\\lib\\site-packages\\Sphinx-1.5.6-py3.6.egg',
 'C:\\Users\\James\\Anaconda3\\lib\\site-packages\\win32',
 'C:\\Users\\James\\Anaconda3\\lib\\site-packages\\win32\\lib',
 'C:\\Users\\James\\Anaconda3\\lib\\site-packages\\Pythonwin',
 'C:\\Users\\James\\Anaconda3\\lib\\site-packages\\setuptools-27.2.0-py3.6.egg',
 'C:\\Users\\James\\Anaconda3\\lib\\site-packages\\IPython\\extensions',
 'C:\\Users\\James\\.ipython']
```

Modules:

Reloading a Module

Import a Module During a Session

The Python interpreter imports a module only once during a session. This makes things more efficient. Here is an example to show how this works.

Suppose we have the following code in a module named my_module.

```
# This module shows the effect of
#  multiple imports and reload
# saved in C:\Users\James\Anaconda3\lib
print("This code got executed")
```


PROFESSIONAL EDUCATION TRAINING

Executed Only Once

```
Now we see the effect of multiple imports.
import my_module
# This code got executed
import my_module
import my_module
```

We can see that our code got executed only once. This goes to say that our module was imported only once. Now if our module changed during the course of the program, we would have to reload it. One way to do this is to restart the interpreter. But this does not help much.

.reload()

Python provides a neat way of doing this. We can use the reload() function inside the imp module to reload a module. This is how its done.

```
import imp

import my_module

imp.reload(my_module)

This code got executed

Out[39]: <module 'my_module' from
'C:\\Users\\James\\Anaconda3\\lib\\my_module.py'>
```


PROFESSIONAL EDUCATION TRAINING

Modules:

Names Defined Inside a Module

Names Defined

We can use the dir() function to find out names that are defined inside a module.

For example, we have defined a function add() in the module example that we had in the beginning.

PROFESSIONAL EDUCATION TRAINING

dir()

```
import example; dir(example)
Out[45]:
['__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'add']
```

Underscore

Here, we can see a sorted list of names (along with add). All other names that begin with an underscore are default Python attributes associated with the module (we did not define them ourselves).

```
For example, the __name__ attribute contains the name of
the module.

import example

example.__name__
```


PROFESSIONAL EDUCATION TRAINING

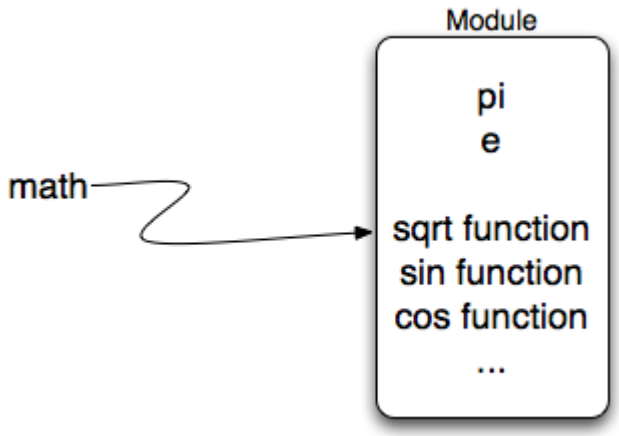
Modules:

Deep Dive into Math Modules

PROFESSIONAL EDUCATION TRAINING

math module

The math module contains the kinds of mathematical functions you would typically find on your calculator and some mathematical constants like pi and e. As we noted above, when we import math, we create a reference to a module object that contains these elements.



PROFESSIONAL EDUCATION TRAINING

Math Functions

Python has a math module that provides most of the familiar mathematical functions.

A module is a file that contains a collection of related functions.

Before we can use the functions in a module, we have to import it with an import statement:

```
import math
```

This statement creates a module object named math.

Dot Notation

To access one of the functions, you have to specify the name of the module and the name of the function, separated by a dot (also known as a period). This format is called dot notation.

```
Module.NameOfFuction
```

```
import math

degrees = 45

radians = degrees / 180.0 * math.pi

math.sin(radians)

0.707106781187

math.sqrt(2) / 2.0

0.707106781187
```


PROFESSIONAL EDUCATION TRAINING

Composition

One of the most useful features of programming languages is their ability to take small building blocks and compose them

```
x = math.sin(degrees / 360.0 * 2 * math.pi)
```

```
0.7071067811865476
```

```
x = math.exp(math.log(x+1))
```


PROFESSIONAL EDUCATION TRAINING

Python Built-in Functions

abs()	divmod()	input()	open()	staticmethod()
all()	enumerate()	int()	ord()	str()
any()	eval()	isinstance()	pow()	sum()
basestring()	execfile()	issubclass()	print()	super()
bin()	file()	iter()	property()	tuple()
bool()	filter()	len()	range()	type()
bytearray()	float()	list()	raw_input()	unichr()
callable()	format()	locals()	reduce()	unicode()
chr()	frozenset()	long()	reload()	vars()
classmethod()	getattr()	map()	repr()	xrange()
cmp()	globals()	max()	reversed()	zip()
compile()	hasattr()	memoryview()	round()	__import__()
complex()	hash()	min()	set()	
delattr()	help()	next()	setattr()	
dict()	hex()	object()	slice()	
dir()	id()	oct()	sorted()	

PROFESSIONAL EDUCATION TRAINING

Modules:

Mathematical Functions

PROFESSIONAL EDUCATION TRAINING

List of Math Functions

Function	Description
abs(x)	The absolute value of x: the (positive) distance between x and zero.
ceil(x)	The ceiling of x: the smallest integer not less than x
cmp(x, y)	-1 if x < y, 0 if x == y, or 1 if x > y
exp(x)	The exponential of x: e^x
fabs(x)	The absolute value of x.
floor(x)	The floor of x: the largest integer not greater than x
log(x)	The natural logarithm of x, for x> 0
log10(x)	The base-10 logarithm of x for x> 0.

PROFESSIONAL EDUCATION TRAINING

List of Math Functions

Function	Description
max(x1, x2,...)	The largest of its arguments: the value closest to positive infinity
min(x1, x2,...)	The smallest of its arguments: the value closest to negative infinity
modf(x)	The fractional and integer parts of x in a two-item tuple. Both parts have the same sign as x. The integer part is returned as a float.
pow(x, y)	The value of x**y.
round(x [,n])	**x** rounded to n digits from the decimal point. Python rounds away from zero as a tie-breaker: round(0.5) is 1.0 and round(-0.5) is -1.0.
sqrt(x)	The square root of x for x > 0

PROFESSIONAL EDUCATION TRAINING

abs()

The method abs() returns absolute value of x - the (positive) distance between x and zero.

```
print ("abs(-45) : ", abs(-45))
print ("abs(100.12) : ", abs(100.12))
```

```
In [48]: print ("abs(-45) : ", abs(-45))
    ...: print ("abs(100.12) : ", abs(100.12))
    ...:
abs(-45) :  45
abs(100.12) :  100.12
```


PROFESSIONAL EDUCATION TRAINING

ceil()

The method ceil() returns ceiling value of x - the smallest integer not less than x. This function is not accessible directly, so we need to import math module and then we need to call this function using math static object.

```
import math   # This will import math module

print ("math.ceil(-45.17) : ", math.ceil(-45.17))

print ("math.ceil(100.12) : ", math.ceil(100.12))

print ("math.ceil(100.72) : ", math.ceil(100.72))
```

```
In [49]: import math   # This will import math module
    ...: print ("math.ceil(-45.17) : ", math.ceil(-45.17))
    ...: print ("math.ceil(100.12) : ", math.ceil(100.12))
    ...: print ("math.ceil(100.72) : ", math.ceil(100.72))
    ...:
math.ceil(-45.17) :  -45
math.ceil(100.12) :  101
math.ceil(100.72) :  101
```


PROFESSIONAL EDUCATION TRAINING

exp()

The method exp() returns exponential of x: ex. This function is not accessible directly, so we need to import math module and then we need to call this function using math static object.

```
import math    # This will import math module

print ("math.exp(-1) : ", math.exp(-1))

print ("math.exp(0) : ", math.exp(0))

print ("math.exp(1) : ", math.exp(1))
```

```
In [54]: import math   # This will import math module
    ...: print ("math.exp(-1) : ", math.exp(-1))
    ...: print ("math.exp(0) : ", math.exp(0))
    ...: print ("math.exp(1) : ", math.exp(1))
    ...:
math.exp(-1) :  0.36787944117144233
math.exp(0) :  1.0
math.exp(1) :  2.718281828459045
```

fabs()

The method fabs() returns the absolute value of x. This function is not accessible directly, so we need to import math module and then we need to call this function using math static object.

The difference is that math.fabs(number) will always return a floating point number even if the argument is integer, whereas abs() will return a floating point or an integer depending upon the argument.

```
print ("math.fabs(1) : ", math.fabs(1))

print ("math.fabs(-1.1) : ", math.fabs(-1.1))
```

```
In [55]: print ("math.fabs(1) : ", math.fabs(1))
    ...: print ("math.fabs(-1.1) : ", math.fabs(-1.1))
    ...:
math.fabs(1) :  1.0
math.fabs(-1.1) :  1.1
```


PROFESSIONAL EDUCATION TRAINING

Random Number Functions

Function	Description
choice(seq)	A random item from a list, tuple, or string.
randrange ([start,] stop [,step])	A randomly selected element from range(start, stop, step)
random()	A random float r, such that 0 is less than or equal to r and r is less than 1
seed([x])	Sets the integer starting value used in generating random numbers. Call this function before calling any other random module function. Returns None.
shuffle(lst)	Randomizes the items of a list in place. Returns None.
uniform(x, y)	A random float r, such that x is less than or equal to r and r is less than y

PROFESSIONAL EDUCATION TRAINING

Trigonometric Functions

Function	Description
acos(x)	Return the arc cosine of x, in radians.
asin(x)	Return the arc sine of x, in radians.
atan(x)	Return the arc tangent of x, in radians.
atan2(y, x)	Return atan(y / x), in radians.
cos(x)	Return the cosine of x radians.
hypot(x, y)	Return the Euclidean norm, sqrt(x*x + y*y).
sin(x)	Return the sine of x radians.
tan(x)	Return the tangent of x radians.
degrees(x)	Converts angle x from radians to degrees.
radians(x)	Converts angle x from degrees to radians.

PROFESSIONAL EDUCATION TRAINING

Number Type Conversion

Type int(x) to convert x to a plain integer.

Type long(x) to convert x to a long integer.

Type float(x) to convert x to a floating-point number.

Type complex(x) to convert x to a complex number with real part x and imaginary part zero.

Type complex(x, y) to convert x and y to a complex number with real part x and imaginary part y. x and y are numeric expressions

PROFESSIONAL EDUCATION TRAINING

Modules:

Matrix Computation

.array()

```
import numpy as np
x = np.array([1,5,2])
y = np.array([7,4,1])
x + y
array([8, 9, 3])
x * y
array([ 7, 20,  2])
```


PROFESSIONAL EDUCATION TRAINING

.array()

```
x - y
array([-6,  1,  1])
x / y
array([0, 1, 2])
x % y
array([1, 1, 0])
x = np.array([1,2,3])
y = np.array([-7,8,9])
np.dot(x,y)
36
```


PROFESSIONAL EDUCATION TRAINING

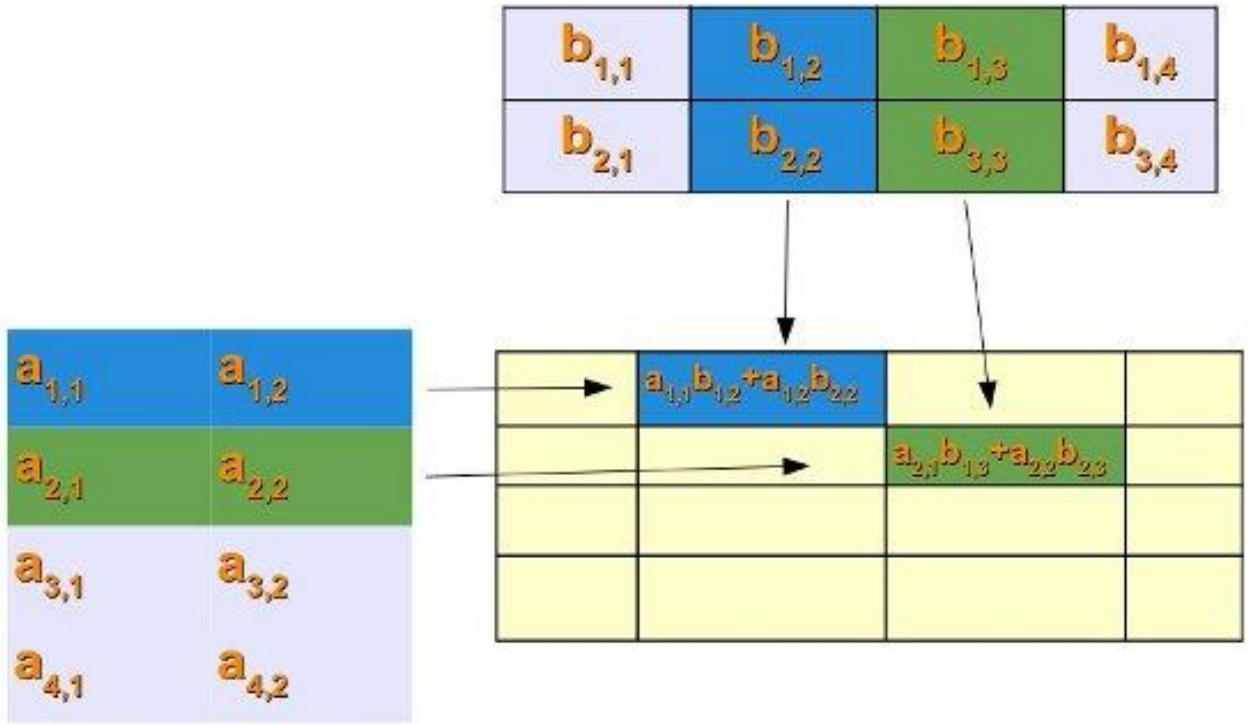
.array()

```
x = np.array((2,3), (3, 5))
y = np.array((1,2), (5, -1))
x * y
array([[ 2,  6],
       [15, -5]])
```


PROFESSIONAL EDUCATION TRAINING

.matrix()

```
x = np.matrix(((2,3), (3, 5)))
y = np.matrix( ((1,2), (5, -1)) )
x * y
matrix([[17,  1],
        [28,  1]])
```



PROFESSIONAL EDUCATION TRAINING

.shape

```
import numpy as np
x = np.matrix('1 2; 3 4')
matrix([[1, 2],
        [3, 4]])

x.shape
Out[63]: (2, 2)
```


PROFESSIONAL EDUCATION TRAINING

.transpose()

```
a = np.array([[1, 2], [3, 4]])
a
array([[1, 2],
       [3, 4]])
a.transpose()
array([[1, 3],
       [2, 4]])
```


PROFESSIONAL EDUCATION TRAINING

Modules:

Time and Date

Date

The datetime class allows us to get the current date and time

```
#The import statement gives us access to
#the functionality of the datetime class
import datetime
#today is a function that returns today's date
print (datetime.date.today())
```

```
In [26]: import datetime
    ...: #today is a function that returns today's date
    ...: print (datetime.date.today())
    ...:
2018-04-09
```


PROFESSIONAL EDUCATION TRAINING

Date

```
import datetime

#store the value in a variable called currentDate
currentDate = datetime.date.today()
print (currentDate)
```


PROFESSIONAL EDUCATION TRAINING

You can access different parts of the date

```
import datetime
currentDate = datetime.date.today()
print (currentDate)
print (currentDate.year)
print (currentDate.month)
print (currentDate.day)
```

```
In [27]: import datetime
    ...: currentDate = datetime.date.today()
    ...: print (currentDate)
    ...: print (currentDate.year)
    ...: print (currentDate.month)
    ...: print (currentDate.day)
    ...:
2018-04-09
2018
4
9
```


PROFESSIONAL EDUCATION TRAINING

Modules:

Formatting Date

Formatting Date

Welcome to one of the things that drives programmers insane!
Different countries and different users like different date formats, often the default isn't what you need
There is always a way to handle it, but it will take a little time and extra code
The default format is `YYYY-MM-DD`

PROFESSIONAL EDUCATION TRAINING

In Python we use strftime to format dates

```
import datetime
currentDate = datetime.date.today()
#strftime allows you to specify the date format
print (currentDate.strftime('%d %b,%Y'))

# %d is the day of the month
# %b is the abbreviation for the month
# %Y is the 4 digit year
```

```
In [28]: import datetime
    ...: currentDate = datetime.date.today()
    ...: #strftime allows you to specify the date format
    ...: print (currentDate.strftime('%d %b,%Y'))
    ...:
09 Apr,2018
```


PROFESSIONAL EDUCATION TRAINING

Formatting Date

Here's a few more you may find useful

- `%b` is the month abbreviation
- `%B` is the full month name
- `%y` is two digit year
- `%a` is the day of the week abbreviated
- `%A` is the day of the week

For a full list visit strftime.org

PROFESSIONAL EDUCATION TRAINING

Formatting Date

"Please attend our event Sunday, July 20 in the year 1997"

```
import datetime
currentDate = datetime.date.today()
#strftime allows you to specify the date format
print (currentDate.strftime
('Please attend our event %A, %B %d in the year %Y'))
```

```
In [29]: import datetime
    ...: currentDate = datetime.date.today()
    ...: #strftime allows you to specify the date format
    ...: print (currentDate.strftime
    ...: ('Please attend our event %A, %B %d in the year %Y'))
    ...:
Please attend our event Monday, April 09 in the year 2018
```


PROFESSIONAL EDUCATION TRAINING

Formatting Date

In programmer speak we call that localization
Did I mention working with dates can be challenging?
By default the program uses the language of the machine where it is running
But… since you can't always rely on computer settings it is possible to force Python to use a particular language
It just takes more time and more code. We won't go into that now, but if you need to do it check out the babel Python library http://babel.pocoo.org/

Formatting Date

```
import datetime
birthday = input ("What is your birthday? ")
birthdate = datetime.datetime.strptime(birthday,"%m/%d/%
Y").date()
#why did we list datetime twice?
#because we are calling the strptime function
#which is part of the datetime class
#which is in the datetime module
print ("Ur birth month is " + birthdate.strftime('%B'))
```


PROFESSIONAL EDUCATION TRAINING

Formatting Date

```
In [38]: import datetime
    ...: birthday = input ("What is your birthday? ")
    ...: birthdate = datetime.datetime.strptime(birthday,"%m/%d/%Y").date()
    ...: #why did we list datetime twice?
    ...: #because we are calling the strptime function
    ...: #which is part of the datetime class
    ...: #which is in the datetime module
    ...: print ("Ur birth month is " + birthdate.strftime('%B'))
    ...:

What is your birthday? 8/8/1988
Ur birth month is August
```


PROFESSIONAL EDUCATION TRAINING

Formatting Date

But what if the user doesn't enter the date in the format I specify in strptime?

```
birthdate = datetime.datetime.strptime(birthday,"%m/%d/%
Y")
# Your code will crash so…
# Tell the user the date format you want
birthday = input ("What is your birthday? (mm/dd/yyyy)
")
# Add error handling, which we will cover in a later
module
```


PROFESSIONAL EDUCATION TRAINING

Formatting Date

```
# You can create a countdown to say
# how many days until a big event or holiday

nextBirthday = datetime.datetime.strptime('08/08/2018','
%m/%d/%Y').date()
currentDate = datetime.date.today()
#If you subtract two dates you get back the number of
days
#between those dates
difference = nextBirthday - currentDate
print (difference.days)
```


PROFESSIONAL EDUCATION TRAINING

Formatting Date

```
In [40]: nextBirthday = datetime.datetime.strptime('08/08/2018','%m/%d/%Y').date()
    ...: currentDate = datetime.date.today()
    ...: #If you subtract two dates you get back the number of days
    ...: #between those dates
    ...: difference = nextBirthday - currentDate
    ...: print (difference.days)
    ...:
121
```


PROFESSIONAL EDUCATION TRAINING

Modules:

Working with Time

Formatting Time

It is called Datetime, so yes, it can store times.

```
import datetime
currentTime = datetime.datetime.now()
print (currentTime)
print (currentTime.hour)
print (currentTime.minute)
print (currentTime.second)
```

```
In [41]: import datetime
    ...: currentTime = datetime.datetime.now()
    ...: print (currentTime)
    ...: print (currentTime.hour)
    ...: print (currentTime.minute)
    ...: print (currentTime.second)
    ...:
2018-04-09 15:03:04.328814
15
3
4
```

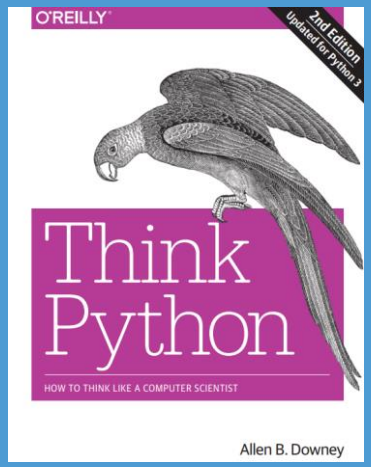

PROFESSIONAL EDUCATION TRAINING

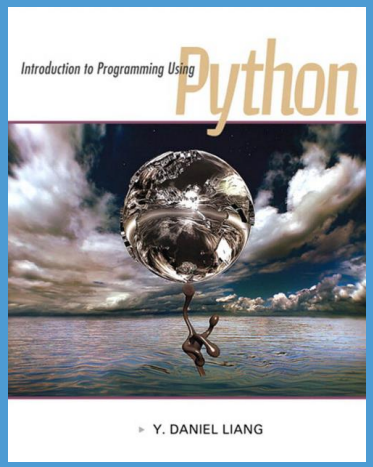
Formatting Time

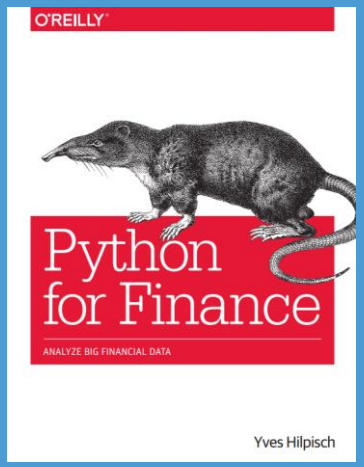
```
import datetime
currentTime = datetime.datetime.now()
print (datetime.datetime.strftime(currentTime,'%H:%M'))

# %H  Hours (24 hr clock)
# %I  Hours (12 hr clock)
# %p  AM or PM
# %m  Minutes
# %S  Seconds
```

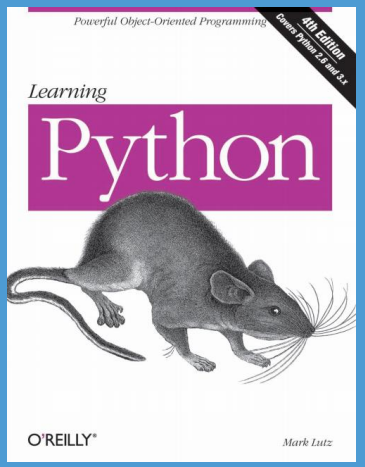
```
In [45]: import datetime
    ...: currentTime = datetime.datetime.now()
    ...: print (datetime.datetime.strftime(currentTime,'%H:%M'))
    ...:
15:20
```

References


O'REILLY
2nd Edition
Updated for Python 3
Think Python
HOW TO THINK LIKE A COMPUTER SCIENTIST
Allen B. Downey


Introduction to Programming Using Python
Y. DANIEL LIANG


O'REILLY
Python for Finance
ANALYZE BIG FINANCIAL DATA
Yves Hilpisch


Powerful Object-Oriented Programming
4th Edition
Learning
Python
O'REILLY
Mark Lutz