CS303E: Elements of Computers and Programming
More Simple Python

Dr. Bill Young
Department of Computer Science
University of Texas at Austin

Last updated: April 12, 2021 at 09:13

Common Python Functions

A **function** is a group of statements that performs a specific task. You'll be writing your own functions soon, but Python also provides many functions for your use.

Function	Description
abs(x)	Return the absolute value
max(x1, x2, ...)	Return the largest arg
min(x1, x2, ...)	Return the smallest arg
pow(a, b)	Return a^b, same as a ** b
round(x)	Return integer nearest x; rounds to even
round(x, b)	Returns the float rounded to b places after the decimal

Common Functions

```
>>> abs( 2 )
2
>>> abs( -2 )              # absolute value
2
>>> max( 1, 4, -3, 6 )
6
>>> min( 1, 4, -3, 6 )
-3
>>> pow( 2, 10 )           # same as 2**10
1024
>>> pow( 2, -2 )           # same as 2**(-2)
0.25
>>> round( 5.4 )
5
>>> round( 5.5 )           # round to even
6
>>> round( 6.5 )           # round to even
6
>>> round( 5.466, 2 )      # set precision
5.47
```

Python Libraries

There are also many available functions that are not in the Python "core" language. These are available in **libraries**.

os — interact with the operating system (e.g., change directory)

math — access special math functions such as log(), sin(), sqrt(), pi

random — random number generation

datetime — clock and calendar functions

To use the functions from a library you have to import it.

Import Examples

```
>>> import os              # import module os
>>> os.name                # what's my OS?
'posix'
>>> os.getcwd()            # get current working directory
'/u/byoung/cs303e/slides'
>>> import random          # import module random
>>> random.random()        # generate random value
0.36552104405513963        # between 0 and 1
>>> random.random()        # do it again
0.7465680663361102
>>> import math            # import module math
>>> math.sqrt( 1000 )      # square root of 1000
31.622776601683793
>>> math.pi                # approximation of pi
3.141592653589793
import datetime            # import module datetime
>>> print(datetime.datetime.now())  # current time
2020-07-30 14:45:42.905190
```

Some Functions in the `math` Library

`floor(x)`	returns the largest integer no bigger than x
`ceil(x)`	returns the smallest integer no less than x
`exp(x)`	exponential function e^x
`log(x)`	natural logarithm (log to the base e of x)
`log(x, b)`	log to the base b of x
`sqrt(x)`	square root of x

Trigonometric functions, including:

`sin(x)`	sine of x
`asin(x)`	arcsine (inverse sine) of x
`degrees(x)`	convert angle x from radians to degrees
`radians(x)`	convert angle x from degrees to radians

Functions from the `math` Library

```
>>> import math
>>> math.floor( 3.2 )
3
>>> math.ceil( 3.2 )
4
>>> math.exp( 2 )          # e ** 2
7.38905609893065
>>> math.log( 7.389 )      # log base e
1.9999924078065106
>>> math.log( 1024, 2 )    # log base 2
10.0
>>> math.sqrt( 1024 )
32.0
>>> math.sin( math.pi )
1.2246467991473532e-16
>>> math.sin( 90 )
0.8939966636005579
>>> math.degrees( math.pi )  # pi radians is 180 deg.
180.0
>>> math.radians( 180 )      # 180 deg. is pi radians
3.141592653589793
```

Using math functions

In file `ComputeAngles.py`:

```
import math

"""Enter the three vertices of a triangle.  Compute and
display the three angles."""

x1, y1, x2, y2, x3, y3 = eval(input("Enter three points: "))

# This computes the lengths of the three sides:
a = math.sqrt((x2 - x3) ** 2 + (y2 - y3) ** 2)
b = math.sqrt((x1 - x3) ** 2 + (y1 - y3) ** 2)
c = math.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)
print("The three sides have lengths: ", round(a, 2), \
      round(b, 2), round(c, 2) )

# This computes the three angles:
A = math.degrees(math.acos((a**2 - b**2 - c**2) / (-2*b*c)))
B = math.degrees(math.acos((b**2 - a**2 - c**2) / (-2*a*c)))
C = math.degrees(math.acos((c**2 - b**2 - a**2) / (-2*a*b)))

print("The three angles are ", round(A * 100) / 100.0,
    round(B * 100) / 100.0, round(C * 100) / 100.0)
```

Executing the Program

```
> python ComputeAngles.py
Enter three points: 1, 1, 6.5, 1, 6.5, 2.5
The three sides have lengths:  1.5 5.7 5.5
The three angles are   15.26 90.0 74.74
```

Note that the string "Enter three points:" was printed by Python. What follows on that line was typed by the user.

Random Numbers

Several useful functions are defined in the `random` module.

`randint(a,b)` : generate a random integer between a and b, inclusively.

`randrange(a, b)` : generate a random integer between a and b-1, inclusively.

`random()` : generate a float in the range $(0 \ldots 1)$.

You need to specify the module, even after you import it.

Random Numbers: Examples

```
>>> import random
>>> random.randint(0, 9)        # same as randrange(0, 10)
8
>>> random.randint(0, 9)
3
>>> random.randrange(0, 10)    # same as randint(0, 9)
2
>>> random.randrange(0, 10)
0
>>> random.randrange(0, 10)
3
>>> random.random()
0.689013943338249
>>> random.random()
0.5466061134029843
```

It's often useful to generate random values to test your programs or to perform scientific experiments.

Aside on Import

There are several different ways to use `import`.

```
>>> import random               # imports module, not names
>>> random()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'module' object is not callable
>>> random.random()
0.46714522525882873
>>> from random import random  # import name random
>>> random()
0.9893720304830842
>>> randint(0, 9)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'randint' is not defined
>>> from random import *       # import all names
>>> randint(0, 9)
5
```

Let's Take a Break

Strings and Characters

A **string** is a sequence of characters. Python treats strings and characters in the same way.

```
letter = 'A'        # same as letter = "A"
numChar = '4'
msg = "Good morning"
```

(Many) characters are represented in memory by binary strings, called the ASCII (American Standard Code for Information Interchange) encoding.

Strings and Characters

A string is represented in memory by a sequence of ASCII character codes. So manipulating characters really means manipulating these numbers in memory.

...	...
...	...
2000	01001010	Encoding for character 'J'
2001	01100001	Encoding for character 'a'
2002	01110110	Encoding for character 'v'
2003	01100001	Encoding for character 'a'
...	...
...	...

Note that a string is *probably* stored internally as a pointer to the first character and a length.

ASCII

The following is part of the ASCII (American Standard Code for Information Interchange) representation for characters.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
32		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
48	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
96	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w	x	y	z	{	—	}		

The standard ASCII table defines 128 character codes (from 0 to 127), of which, the first 32 are control codes (non-printable), and the remaining 96 character codes are representable characters.

Unicode

ASCII codes are only 7 bits (some are extended to 8 bits). 7 bits only allows 128 characters. *There are many more characters than that in the world.*

Unicode is an extension to ASCII that uses multiple bytes for character encodings. With Unicode you can have Chinese characters, Hebrew characters, Greek characters, etc.

Unicode was defined such that ASCII is a subset. So Unicode readers recognize ASCII.

Operating on Characters

Notice that:

- The lowercase letters have consecutive ASCII values (97...122); so do the uppercase letters (65...90).
- The uppercase letters have lower ASCII values than the uppercase letters, so "less" alphabetically.
- There is a difference of 32 between any lowercase letter and the corresponding uppercase letter.

To convert from upper to lower, add 32 to the ASCII value.

To convert from lower to upper, subtract 32 from the ASCII value.

To sort characters/strings, sort their ASCII representations.

ord and chr

Two useful functions for characters:

ord(c) : give the ASCII code for character c; returns a number.

chr(n) : give the character with ASCII code n; returns a character.

```
>>> ord('a')
97
>>> ord('A')
65
>>> diff = (ord('a') - ord('A'))
>>> diff
32
>>> upper = 'R'
>>> lower = chr( ord(upper) + diff ) # upper to lower
>>> lower
'r'
>>> lower = 'm'
>>> upper = chr( ord(lower) - diff ) # lower to upper
>>> upper
'M'
```

Escape Characters

Some special characters wouldn't be easy to include in strings, e.g., single or double quotes.

```
>>> print("He said: "Hello"")
  File "<stdin>", line 1
    print("He said: "Hello"")
                          ^
SyntaxError: invalid syntax
```

What went wrong?

To include these in a string, we need an *escape sequence*.

Escape Sequence	Name	Escape Sequence	Name
\b	backspace	\r	carriage return
\t	tab	\\	backslash
\n	linefeed	\'	single quote
\f	formfeed	\"	double quote

Printing without a Newline

Our previous examples using the `print` function always added a newline at the end.

```
>>> print("abc")
abc
>>>
```

You can print without the implicit newline using the end parameter.

```
>>> print("abc", end = "")
abc>>> print("abc", end = 'xyz\n')
abcxyz
>>>
```

Note that `print(x)` is equivalent to `print(x, end='\n')`.

Printing with Different Separation

By default, `print` separates arguments with a space.

```
>>> print(1, 2, 3, 4)
1 2 3 4
>>> print("abc", "def")
abc def
```

You can override that behavior using the sep parameter.

```
>>> print(1, 2, 3, 4, sep="")
1234
>>> print("abc", "def", sep="xyz")
abcxyzdef
>>> print(8, 5, 2020, sep="/")
8/5/2020
```

Keyboard Input

The `input()` function is used to read data from the user during program execution.

General form:

    input (<prompt string >)

When it's called:

- It prints the "prompt string" to the terminal. This is the message to tell the user to enter some input.
- It waits until the user types something and hits "Enter" or "Return."
- It reads in what the user typed *as a string.*

Input Example

```
>>> input("Enter a number: ")
Enter a number: 32
'32'
>>> numEntered = input("Enter a number: ")
Enter a number: 32
>>> numEntered + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not int
>>> int(numEntered) + 1
33
```

Notice that the error happened because we tried to add a `str` to an integer.

Remember that keyboard input is always read as a `str`. Interpret that string as an integer by using the `int` function.

The "+" operator can be used to concatenate two strings:

```
>>> msg1 = "My name is Bill "
>>> msg2 = "Young"
>>> msg1 + msg2
'My name is Bill Young'
>>> msg3 = "Good night"
>>> msg3 += " and good luck!"
>>> msg3
'Good night and good luck!'
```

Here "+" is actually shorthand for a method (function) associated with the `str` object *class*. More on classes later.

Note that "+" is *overloaded*, meaning that this same syntax is used for multiple purposes. The Python interpreter must figure out what you meant.

```
>>> 123 + 456                       # add two ints
579
>>> 12.3 + 45.6                     # notice approximation
57.900000000000006
>>> "123" + "456"                   # "add" two strings
'123456'
>>> 123 + "456"                     # add int and str
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +:'int' and 'str'
```

Python doesn't natively define "+" to have meaning in the last case, but you could if you wanted to!

Note that `print(x)` automatically coerces `x` to a `str` value.

```
>>> print( 19.2, -20, "abc" )
19.2 -20 abc
```

Often you'd like to print numbers and strings nicely (fixed width, certain precision, right or left justified). Use the `format` function.

General Form:

```
format ( value, format-string )
```

This generates a `str` representation of `value` formatted as indicated by the `format-string`.

Formatting: Floats

To specify a format for a **float**, give a *format string* including:

1. field width: overall width of the string printed
2. precision: number of digits after the decimal point

For example, the format string "10.2f", means to print a float (the "f") right justified in a field of width 10, with 2 digits after the decimal point.

```
>>> format(123.456789, "10.2f")
'    123.46'
>>> format(123456.7891234, "8.3f")
'123456.789'
>>> format(123, "8.3f")
' 123.000'
```

Notice that if the field is not wide enough for the value, it is expanded.

Formatting: Floats

You can also format floats in scientific notation or as percentages.

```
>>> format(123.45678, "10.2e")
'   1.23e+02'
>>> format(123.45678, "10.2%")
' 12345.68%'
>>> format(0.000123, "10.2e")
'   1.23e-04'
>>> format(0.000123, "10.2%")
'     0.01%'
```

And you can *left justify*:

```
>>> format(0.000123, "<10.2%")
'0.01%     '
>>> format(123.45678, "<10.2e")
'1.23e+02  '
```

Formatting: Ints

You can format an integer in decimal, hexadecimal, octal, or binary. You can also specify a *field width*.

```
>>> format(10000, "10d")
'     10000'
>>> format(10000, "10x")
'      2710'
>>> format(10000, "10o")
'     23420'
>>> format(10000, "10b")
'10011100010000'
```

Again, you can left justify:

```
>>> format(10000, "<10d")
'10000     '
```

Formatting: Strings

For strings, you can specify a width. Strings are left justified by default. You can right justify with ">".

```
>>> format("Hello, world!", "20s")
'Hello, world!       '
>>> format("Hello, world!", "5s")
'Hello, world!'
>>> format("Hello, world!", ">20s")
'       Hello, world!'
```

Let's design a program to input names, midterm and final exam grades for three students, compute the average for each and print the results out in a nice table.

In file `ExamExample.py`:

```python
def main():

    """Input the names of three students with two exam
    grades for each, compute their test average, and
    print in table form."""

    # Enter info for Student1
    name1 = input("Enter the first student's name: ")
    midterm1 = int( input("Enter " + name1 + \
        "\'s midterm grade: ") )
    final1 = int( input("Enter " + name1 + \
        "\'s final exam grade: ") )

    < Do the same for the other two students. >
```

```python
    # Now, we print the table:

    # Print header:
    print( "\nName           MT   FN    Avg")
    print( "---------------------------")

    # Student1
    avg1 = (midterm1 + final1) / 2
    print( format(name1, "10s"), format(midterm1, "4d"), \
           format(final1, "4d"), format(avg1, "7.2f") )

    < Do the same for the other two students. >

main()
```

Running Our Program

```
> python ExamExample.py
Enter the first student's name: Charlie
Enter Charlie's midterm grade: 90
Enter Charlie's final exam grade: 75

Enter the second student's name: Susie
Enter Susie's midterm grade: 60
Enter Susie's final exam grade: 80

Enter the third student's name: Frank
Enter Frank's midterm grade: 8
Enter Frank's final exam grade: 77

Name           MT   FN    Avg
---------------------------
Charlie        90   75   82.50
Susie          60   80   70.00
Frank           8   77   42.50
```