02-3 Functions

CSI 500

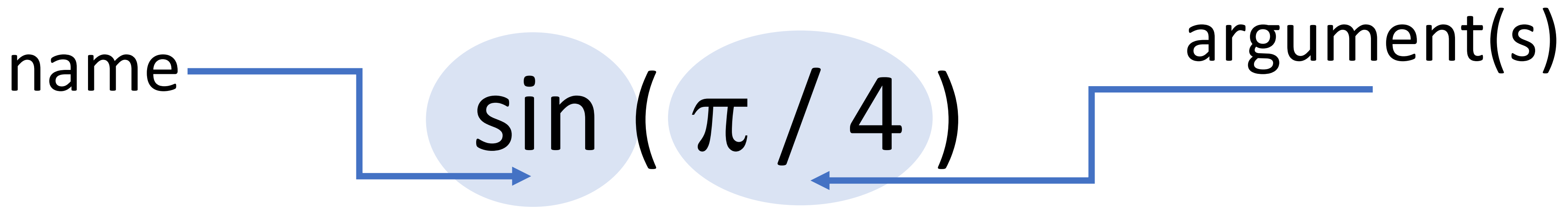
Spring 2018

Course material derived from:
Downey, Allen B. 2012. "Think Python, 2nd Edition". O'Reilly Media Inc., Sebastopol CA.

"How to Think Like a Computer Scientist" by Peter Wentworth, Jeffrey Elkner, Allen B. Downey, and Chris Meyers. Oct 2012
http://openbookproject.net/thinkcs/python/english3e/index.html

Functions

- What is a function?
  - Set of statements used to perform a task
  - Uniquely identified by a **name**
  - May include one or more **arguments** as parameters



Type conversion functions

- Built-in Python functions to convert from one type to another
  - int (arg) : converts strings and floats into integer
  - float(arg) : converts integers and strings into float
  - str(arg) : converts int and float into a string

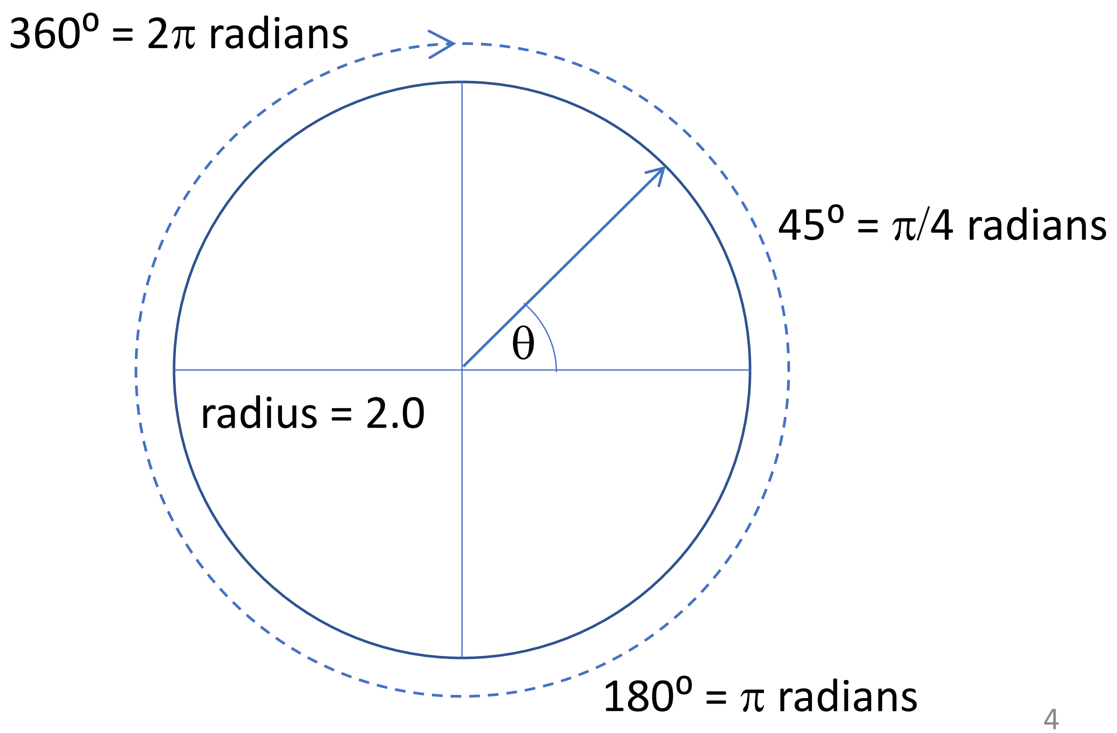
```
int( '32')   # string to int
32
int('hello' )             # can't convert words
ValueError: invalid literal for int(): hello
int( 3.999 )              # float to int (truncated)
3
int( -2.3 ) # float to int (truncated)
-2
```

```
float( 32 )# int to float
32.0
float( '3.1415')        # str to float
3.1415

str( 32 )    # int to str
'32'
str( 3.1415 )           # float to str
'3.1415'
```

Math functions

- Built-in Python module **math** provides standard math library
  - You must import the library to use it, via the **import math** statement
  - Functions may be used individually or combined (composition)

```
import math
radius = 2.0
area = math.pi * radius ** 2
area
12.566370614359172

theta = math.pi / 4.0
math.sin( theta )
0.7071067811865475
```



Adding new functions

- You can create new functions using the def operator
  - Short for "define"
  - Parameters are optional, enclosed in parenthesis
  - Def statement must end with a colon :
  - Blocks of statements are indented

```
def camelot():                        # define function
...           print "We're Knights of the Round Table."
...           print "We dance whene'er we're able."
...

print( camelot )
<function camelot at 0x00000000019E8D68>
type( camelot )
<type 'function'>

camelot()
We're Knights of the Round Table.
We dance when e're we're able.

def repeat_lyrics():  # new function
...           camelot()
...           camelot()

repeat_lyrics()                       # what happens?
```

Parameters and Arguments

- Functions may take one or more **parameters**
  - Values identified in the calling signature
  - Used within the function
- When invoking a function, you supply **arguments**
  - These are the values you want to supply to your function
  - Used when the function is run

parameter

```
def print_twice( message ):
...             print message
...             print message

print_twice( 'hello ')
hello
hello

print_twice( 'spam' )
spam
spam
```

argument

Variables and Parameters are local

- Any parameter names declared in the function signature are **local** to that function only
- Any variables used within the function are **local** to that function only

msg_len OK to use locally

msg_len **not OK** to use globally

```
def print_twice( msg ):
        msg_len = len(msg)
        print "msg_len = " + str(msg_len)
        print msg
        print msg

print_twice("hello")
msg_len = 5
hello
hello
print msg_len

Traceback (most recent call last):
  File "<pyshell#134>", line 1, in <module>
    print msg_len
NameError: name 'msg_len' is not defined
```

Global variables

- Global variables used anywhere
- Often used as
  - default values
  - constants
  - flags indicating conditions
- Use the "global" keyword
  - indicate use of a global variable inside a function
  - however, global lists or dictionaries may be accessed w/o using the "global" keyword

```
count = 0                # count is GLOBAL
def inc():
        count = count + 1
inc()
Traceback (most recent call last):
  File "<pyshell#59>", line 1, in <module>
    inc()
  File "<pyshell#58>", line 2, in inc
    count = count + 1
UnboundLocalError: local variable 'count' referenced
before assignment
def inc():
        global count
        count = count + 1

inc()
count
1

inc()
count
2
```

Fruitful and Void Functions

- Functions that return a useful value are dubbed "fruitful" (this is Downey's terminology; not universally adopted)
  - In interactive mode, a fruitful function will print results to the console
  - In script mode, you need to assign the result to a variable if you want to use it later
- Functions that do something but don't return a meaningful value are called void functions (a similar construct exists in C/C++)
  - For example, when you call "print" it prints, but does not return a meaningful value
  - when you call sort, it sorts the object but does not return a meaningful value
- Functions that don't return a meaningful value return a special Python value called **None** (which is not surprisingly of type "NoneType")

Importing modules

- Python provides a large number of add-on modules with extended capabilities
- You may import an entire module using the **import module** statement
  - Gets all functions and definitions from that module
  - Requires dotted notation to access
- You may import specific parts of a module by using the **from module import item** syntax
  - This avoids dotted notation to access
  - But requires you to know exactly what you want to get from the module beforehand

```
import math
print math
<module 'math' (built-in)>
print math.pi
3.14159265359

print pi
Traceback (most recent call last):
  File "<pyshell#139>", line 1, in <module>
    print pi
NameError: name 'pi' is not defined

from math import pi
print pi
3.14159265359
```

Summary

- Python includes built-in libraries called modules
  - contain global constants
  - contain specialized functions
- You can write your own functions
  - variables declared in a function are local
  - parameters are local
- Python supports global variables
  - often used as flags, constants, or as program controls