Intro to Python

Jan 24th

Slides by M. Stepp, M. Goldstein, M. DiRamio, and S. Shah

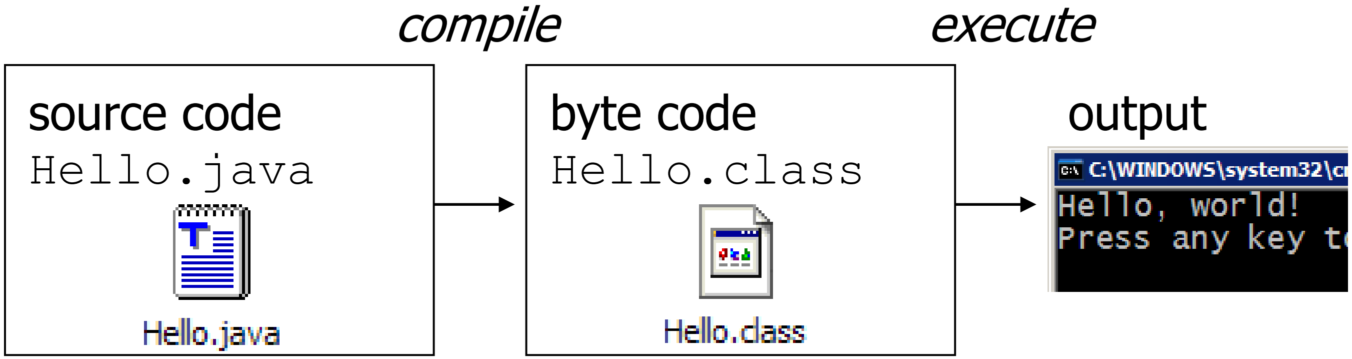
Digital Image Processing

COSC 6380/4393

- **Python and OpenCV Setup**

Compiling and interpreting

- Many languages require you to *compile* (translate) your program into a form that the machine understands.



compile *execute*

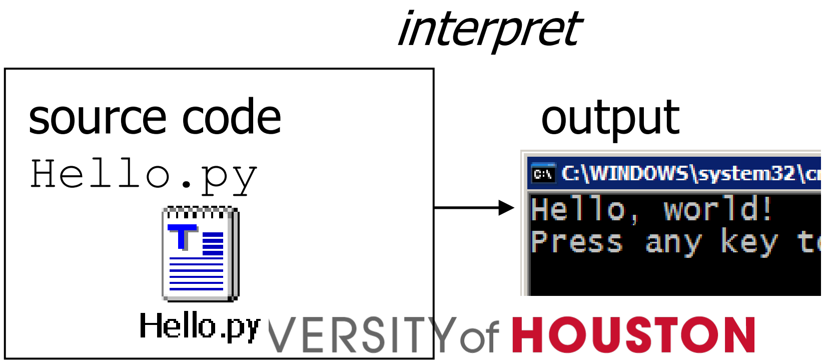
source code
`Hello.java`
Hello.java

byte code
`Hello.class`
Hello.class

output

```
Hello, world!
Press any key t
```

- Python is instead directly *interpreted* into machine instructions.



interpret

source code
`Hello.py`
Hello.py

output

```
Hello, world!
Press any key t
```

The Python Interpreter

- If installed:
  - Open *Command prompt*:

    > python

- Else
  - https://repl.it/languages/python3

Overview

1. Variables and Datatypes
2. Control statements
3. Functions and Modules
4. Matrix Operations

Variables

- Are not declared, just assigned
  - The variable is created the first time you assign it a value

```
>>> x = 7
>>> x
7
```

- Are references to objects
  - Type information is with the object, not the reference
- Everything in Python is an object

reference
object

```
>>> x = 'hello'
>>> x
'hello'
```

- Everything means everything, including <u>functions</u> and <u>classes</u> (more on this later!)
- <u>Data type</u> is a property of the object and not of the variable

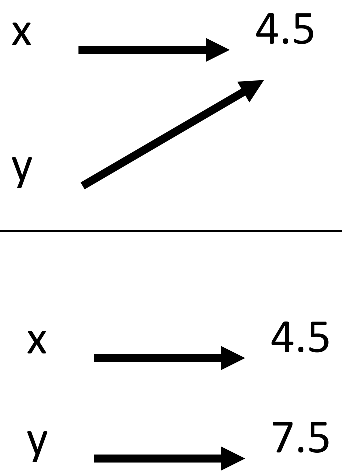
Numbers

- Types: Int, Long, Float, Complex
- Convert between types:
  - int(x) converts x to an integer
  - float(x) converts x to a floating point
- Complex type built into python
  - Same operations are supported as integer and float

```
>>> x = 3 + 2j
>>> y = -1j
>>> x + y
(3+1j)
>>> x * y
(2-3j)
```

Numbers are immutable

```
>>> x = 4.5
>>> y = x
>>> y += 3
>>> x
4.5
>>> y
7.5
```



Data Structures

1. Lists
   - holds an ordered collection of items
2. Tuples
   - similar to lists, they are *immutable*
3. Dictionaries
   - holds key – value pairs

Lists

- Ordered collection of data
- Data can be of different types
- Lists are *mutable*
- Same subset operations as Strings

```
>>> x = [1,'hello', (3 + 2j)]
>>> x
[1, 'hello', (3+2j)]
>>> x[2]
(3+2j)
>>> x[0:2]
[1, 'hello']
```

Lists: Modifying Content

- x[i] = a   reassigns the ith element to the value a
- Since x and y point to the same list object, both are changed
- The method append also modifies the list

```
>>> x = [1,2,3]
>>> y = x
>>> x[1] = 15
>>> x
[1, 15, 3]
>>> y
[1, 15, 3]
>>> x.append(12)
>>> y
[1, 15, 3, 12]
```

Jupyter

Tuples

- Tuples are *immutable* versions of lists
- One strange point is the format to make a tuple with one element:

  ‘,’ is needed to differentiate from the mathematical expression (2)

```
>>> x = (1,2,3)
>>> x[1:]
(2, 3)
>>> y = (2,)
>>> y
(2,)
>>>
```

Jupyter

Dictionaries

- A set of key-value pairs
- Dictionaries are *mutable*

```
>>> d = {1 : 'hello', 'two' : 42, 'blah' : [1,2,3]}
>>> d
{1: 'hello', 'two': 42, 'blah': [1, 2, 3]}
>>> d['blah']
[1, 2, 3]
```

Dictionaries: Add/Modify

- Entries can be changed by assigning to that entry

```
>>> d
{1: 'hello', 'two': 42, 'blah': [1, 2, 3]}
>>> d['two'] = 99
>>> d
{1: 'hello', 'two': 99, 'blah': [1, 2, 3]}
```

- Assigning to a key that does not exist adds an entry

```
>>> d[7] = 'new entry'
>>> d
{1: 'hello', 7: 'new entry', 'two': 99, 'blah': [1, 2, 3]}
```

Dictionaries: Deleting Elements

- The del method deletes an element from a dictionary

```
>>> d
{1: 'hello', 2: 'there', 10: 'world'}
>>> del(d[2])
>>> d
{1: 'hello', 10: 'world'}
```

Jupyter

Copying Dictionaries and Lists

- The built-in **list** function will copy a list
- The dictionary has a method called **copy**

```
>>> l1 = [1]
>>> l2 = list(l1)
>>> l1[0] = 22
>>> l1
[22]
>>> l2
[1]
```

```
>>> d = {1 : 10}
>>> d2 = d.copy()
>>> d[1] = 22
>>> d
{1: 22}
>>> d2
{1: 10}
```

Data Type Summary

- Lists, Tuples, and Dictionaries can store any type (including other lists, tuples, and dictionaries!)
- Only lists and dictionaries are mutable
- All variables are references

Data Type Summary

- Integers: 2323, 3234L
- Floating Point: 32.3, 3.1E2
- Complex: 3 + 2j, 1j
- Lists: l = [1,2,3]
- Tuples: t = (1,2,3)
- Dictionaries: d = {‘hello’ : ‘there’, 2 : 15}

Booleans

- 0 and None are false
- Everything else is true
- True and False are aliases for 1 and 0, respectively

No Braces

- Python uses ***indentation*** instead of braces to determine the scope of expressions
- All lines must be indented the same amount to be part of the scope (or indented more if part of an inner scope)
- This **forces** the programmer to use proper indentation since the indenting is part of the program!

If Statements

```
import math
x = 30
if x <= 15 :
    y = x + 15
elif x <= 30  :
    y = x + 30
else :
    y = x
print("y = ", math.sin(y))
```

```
y = -0.3048106211022167
```

Jupyter

While Loops

```
x = 1
while x < 10 :
    print(x)
    x = x + 1
```

```
1
2
3
4
5
6
7
8
9
```

Loop Control Statements

break	Jumps out of the closest enclosing loop
continue	Jumps to the top of the closest enclosing loop
pass	Does nothing, empty statement placeholder

The Loop Else Clause

- The optional **else** clause runs only if the loop exits normally (not by break)

```
x = 1

while x < 3 :
    print(x)
    x = x + 1
else:
    print(“hello”)

  1
  2
  hello
```

```
x = 1

while x < 5 :
    print(x)
    x = x + 1
    break
else :
    print(“i got here”)

  1
```

For Loops

- For loops: iterating through a list of values

```
for x in [1,7,13,2] :
    print x


1
7
13
2
```

```
for x in range(5) :
    print x


0
1
2
3
4
```

range(n) generates a list of numbers [0,1, …, n-1]

- **For** loops also may have the optional **else** clause

```
for x in range(5):
    print x
    break
else :
    print(“i got here”)


 1
```

Jupyter

Function Basics

```
def max(x,y) :
    if x < y :
        return y
    else :
        return x
```

```
>>>>>> max(3,5)
5
>>> max('hello', 'there')
'there'
>>> max(3, 'hello')
TypeError
```

Functions are first class objects

- Can be assigned to a variable
- Can be passed as a parameter
- Can be returned from a function
- Functions are treated like any other variable in Python, the **def** statement simply assigns a function to a variable

Function names are like any variable

- Functions are objects
- The same reference rules hold for them as for other objects

```
>>> x = 10
>>> x
10
>>> def x () :
...     print(“hello”)
>>> x
<function x at 0x619f0>
>>> x()
hello
>>> x = 'blah'
>>> x
'blah'
```

Functions as Parameters

```
def foo(f, a) :
   return f(a)

def bar(x) :
   return x * x
```

```
>>> foo(bar, 3)
9
```

Note that the function foo takes two parameters and applies the first as a function with the second as its parameter

Functions Inside Functions

- Since they are like any other object, you can have functions inside functions

```
def foo (x,y) :
    def bar (z) :
        return z * 2
    return bar(x) + y
```

```
>>> foo(2,3)
7
```

Functions Returning Functions

```
def foo (x) :
   def bar(y) :
      return x + y
   return bar

# main
f = foo(3)
print f
print f(2)
```

```
5
```

Parameters: Named

- Call by name
- Any positional arguments must come before named ones in a call

```
>>> def foo (a,b,c) :
...     print a, b, c
...
>>> foo(c = 10, a = 2, b = 14)
2 14 10
>>> foo(3, c = 2, b = 19)
3 19 2
```

Parameters: Defaults

- Parameters can be assigned default values
- They are overridden if a parameter is given for them
- The type of the default doesn’t limit the type of a parameter

```
>>> def foo(x = 3) :
...     print x
...
>>> foo()
3
>>> foo(10)
10
>>> foo('hello')
hello
```

Modules: Imports

import mymodule	Brings all elements of mymodule in, but must refer to as mymodule.<elem>
from mymodule import x	Imports x from mymodule right into this namespace
from mymodule import *	Imports all elements of mymodule into this namespace

Math commands

- Python has useful commands for performing calculations.

Command name	**Description**
`abs(`***value***`)`	absolute value
`ceil(`***value***`)`	rounds up
`cos(`***value***`)`	cosine, in radians
`floor(`***value***`)`	rounds down
`log(`***value***`)`	logarithm, base *e*
`log10(`***value***`)`	logarithm, base 10
`max(`***value1***`,` ***value2***`)`	larger of two values
`min(`***value1***`,` ***value2***`)`	smaller of two values
`round(`***value***`)`	nearest whole number
`sin(`***value***`)`	sine, in radians
`sqrt(`***value***`)`	square root

Constant	**Description**
`e`	2.7182818...
`pi`	3.1415926...

- To use many of these commands, you must write the following at the top of your Python program:

```
from math import *
```

Logic

- Many logical expressions use *relational operators*:

Operator	Meaning	Example	Result
`==`	equals	`1 + 1 == 2`	`True`
`!=`	does not equal	`3.2 != 2.5`	`True`
`<`	less than	`10 < 5`	`False`
`>`	greater than	`10 > 5`	`True`
`<=`	less than or equal to	`126 <= 100`	`False`
`>=`	greater than or equal to	`5.0 >= 5.0`	`True`

- Logical expressions can be combined with *logical operators*:

Operator	Example	Result
`and`	`9 != 6 and 2 < 3`	`True`
`or`	`2 == 3 or -1 < 5`	`True`
`not`	`not 7 > 0`	`False`

Exercise: Evaluate the quadratic equation ax^2 + bx + c = 0 for a given *a*, *b*, and *c*.

Exercise: Evaluate the quadratic equation ax^2 + bx + c = 0 for a given *a*, *b*, and *c*.

```
from math import *
Print("Hello! This is my quadratic equation program." )
a = input("What is a? ")
b = input("What is b? ")
c = input("What is c? ")
root1 = (-b + sqrt(b ** 2 - 4 * a * c)) / (2 * a)
root2 = (-b - sqrt(b ** 2 - 4 * a * c)) / (2 * a)
Print("The roots are", root1, "and", root2)
```

Exercise: How would we print the "99 Bottles of Beer" song?

Exercise: How would we print the "99 Bottles of Beer" song?

```
maxBottles = 99
for bottles in range(maxBottles, 0, -1):
        print(bottles, "bottles of beer on the wall" )
        print(bottles, "bottles of beer“)
        print("You take one down“)
        print("Pass it around“)
        print(bottles - 1, "bottles of beer on the wall" print "“)

print("Oh no, we're out of beer.“)
```

Exercise: Write a Python program that computes the factorial of an integer.

Exercise: Write a Python program that computes the factorial of an integer.

```
n = input("Factorial of what number? ")
fact = 1
for i in range(1, n + 1):
        fact = fact * i

print "The factorial is", fact
```