February 3rd, 2018
TA: Kelly Ryu

Lecture 2 Computer Programming in Python (Accelerated)

Overview of Topics Covered in 5 Lectures

- **Lecture 2:** Datatypes, Operators, Control Flow, Basic I/O
- **Lecture 3:** Writing Functions
- **Lecture 4:** Object-Oriented Programming
- **Lecture 5:** Advanced I/O
- **Lecture 6: NumPy and Matplotlib**

Today:

A. **Datatypes**
B. **Operators and Methods/Functions**
   (https://docs.python.org/2/library for more methods)
C. **Control Flow**
D. **Basic I/O**
E. **More methods (if time)**
F. **(FYI) Big terms in Python**

A. Datatypes

Datatypes: a set of values from which an expression can take its value and operations that can be used on them

- Integers & the set of manipulations that can be used on them, such as addition, subtraction, etc. → class `int`
- Floating point numbers & the set of manipulations that can be used on them, such as addition, subtraction, etc. → class `float`
- Booleans & negation, and/or operations → class `bool`
- Strings & concatenation, indexing → class `str`
- Lists & sorting, indexing, searching → class `list`
- Datatypes have nothing to do with the computer or programming language: they are abstract categorizations of information ← → The representation in computers is class

Assigning variables in Python

- Unlike Java, no need to declare variable types (`int`, `bool`, etc.)
- No need for colons
- Written in console:

```
In[1]: a=3
In[2]: a
Out[2]: 3
```

B. Operators and Methods/Functions

1. Integers, Doubles, Floats (`int`, `double`, `float`)>

1) Basic operators

Arithmetic operators:

+ (addition), - (subtraction), / (division), // (floor division), %(modulus), **(exponent), *(multiplication)

** For / (regular division) on integers, the result is the same as floor division. So you must convert at least one integer to a float to do exact division on integers.

```
>>> 7/3
2
>>> 7//3
2
>>> 7/3.0
2.3333333333333335
>>> 7//3.0
2.0
```

Comparison operators:

== (equal), != (unequal), >, <, <=, >=

2) Functions in `math` module

First, the keyword `import`: brings functions that are not built-in (e.g. math module)

Eg) **notice how we first imported math, then also specified the math module again when calling the constants and methods from this module using "math-dot"

** details about `print` come later

```
import math
print 'some constants in Math module'
print 'pi: {:.4f}'.format(math.pi) #print value of pi up to 4 decimal points

print 'some common functions'
print 'square root of 3: {:.4f}'.format(math.sqrt(3))
```

- again, we are formatting the square root of 3 up to 4 decimal points
- we are calling the `format()` method on the parameter `math.sqrt(3)`
- `sqrt()` is also a method in the `math` module; the parameter `3` is used

```
print 'square root of 2: {:.4f}'.format(math.pow(2,.5))
```

- `pow()` is a method in the `math` module; the parameters `2` and `.5` are used

```
print 'sin of pi/2: {:.4f}'.format(math.sin(math.pi/2))
```

More `math` functions here: https://docs.python.org/2.7/library/math.html

3) Casting between `float` and `int` (built-in casting function)

```
In: int(7.99)
Out: 7

In: float(7)
Out: 7.0
```

4) Functions in `random` module

```
In: import random
In: int(random.random()*3+1)
Out: 0.5
```

- `random()` function produces a random number between 0 and 1 (equal distribution between 0~1/3, 1/3~2/3, and 2/3~1)
- *3 → Multiply by 3 : equal distribution between 0~1, 1~2, 2~3
- +1 → equal distribution of 1~2, 2~3, 3~4
- take an `int` of that → equal percentage of 1,2,3 = rock, paper, scissors
- we are calling the function `random()` in the `random` module; no parameters this time = nothing in the parenthesis after `random`

Or a more convenient function is `randint(`*`start,end`*`)`

```
In: import random
In: random.randint(0,3)
Out: 1   ## any random integer that is 0≤x≤3 (note! inclusive of boundaries)
```

2. **Booleans:** Their literals are `True or False`

```
In: B = True
In: B
Out: True

In: a=3
In: a>2
Out: True
```

`In: C = a>2` ('=' is an operator; the LHS, 'C,' is a variable and RHS, 'a>2,' is a Boolean expression that is to be assessed true or false)

```
In: C
Out: True
```

`In: D = a==2` ('==' means the relation 'equals', thus the question whether it's equal or not)

```
In: D
Out: False
```
(The Boolean that says a is equal to 2 is false)

- Use keywords `and` and `or` to evaluate multiple Booleans together

3. Strings

** a sequence of characters
** expressed inside quotations (either single or double: doesn't matter which)
** abbreviated **str,** which indicates the class of strings

```
In: a = 'happy'
In: a
Out: 'happy'

In: a= "'happy'"
In: a
Out: "'happy'" (single quotes is included in the string in this case)
```

What are the operations you can do with strings?

1) Concatenation

```
In: a= 'happy'+'joy'
In: a
Out: 'happyjoy'
```

2) Array notation: access a certain part of a string

BUT remember that you start counting from 0 (so the first component is index 0)

```
In: a[1] (find the index 1 of the 'a' string)
Out: 'h' ('h' is the second component of the string 'a', you count the first letter starting from number 0)

In: a[0]
Out: "'" (It uses the double quotes to indicate it is a string because the component we found is a single quotation mark)
```

4. LISTS: ordered set of stuff (any mix of data types) or maybe it could be an empty list

```
l = [4, 1, 'happy', True]
l2 = [4, 3, 'sad', 1]
```

- Access elements of a list

```
In: l[0]
Out: 4
```

- Modify/Set an element in a list

```
In: l[3]='munday'
In: l
Out: [4,1,'happy','munday']
```

- Find length

```
In: l.len()
Out: 4
```

C. Control Flow

1. if (conditional)

```
if (Boolean expression1):
   things to execute if expression1 is true

elif (Boolean expression2):
   things to execute if 1 is false and 2 is true

else:
   things to execute if 1 and 2 are false
```

Eg)

```
a=20
if(a<10):
    b=12
else:
    b=3
print b
```

(Since the Boolean a<10 is False, b=12 Is not executed and the 'else' case, which is b=3, is executed.)

Eg)

Case1>>

```
x=3
if(x==3):
    print 'happy'
print 'sad'
```

Case 2>>

```
if(x==3):
    print 'happy'
else:
    print 'sad'
```

** In case 1, 'happy' and 'sad' would be both printed because print('sad') function is always carried out. In case 2, only 'happy' would be printed… 'sad' would be printed only when x is not equal to 3. MAKE SURE YOU UNDERSTAND THE CONCEPT OF 'ELSE'!

** Also note that unlike Java, where if/else clauses are separated clearly with brackets, the only way to tell when the if clause ends is through indentation. In Case 1, since `print 'happy'` is indented, it is included in the if clause. Since `print 'happy'` is not indented, it is not part of the if clause and the if clause ends just before it.

2. While (loop)

```
while(Boolean expression):
     stuff to do until expression becomes false
```

- If the Boolean statement is true, we keep running
- Sometimes it results in an infinite loop
- Thus it's important to include some change of variable in the while loop so that eventually, the Boolean expression becomes false

Eg)

```
X=5
while(X>3):
     print 'happy'
     X=X-1
```

For the first loop, the program will print 'happy' and the variable becomes 4
For the second loop, the program prints 'happy' and the variable becomes 3
Then X is no longer larger than 3 and the program exits the loop

3. for (loop): logically equivalent to a while loop (simply a matter of convenience)

```
In: l = [4,1,'happy','munday']
In: for item in l:
     print item
Out:
4
1
happy
munday
```

**difference in convenience (for and while)

- UNTIL: use "while" loop (keep repeating UNTIL that while loop is false)
- If I just want to repeat it for everything, use "for"

** `range` is useful in for loops

```
In: for i in range(10):
          print i
```

```
Out:
0
1
2
3
4
5
6
7
8
9
```

D. Basic I/O

1. Input

variable name = raw_input("*string to print before user enters*")

- waits for the user to input information and sends the string literal back to the variable on the LHS
- To make the input info into a different data-type other than a string, we must put `int()`, `float()` or `bool()` around the input as below
- Eg. if the user types "15" for the input, we know 15 is an integer, but it is in the `str` class for the computer

Eg)

```
name=raw_input("What's your name dude?: ")
fav_color=raw_input("What's your favorite color?: ")
fav_number=int(raw_input("What's your favorite number?: "))
fav_float=float(raw_input("What's your favorite non-integral
                          number?: "))
```

2. Output

print "*anything you want to print*"

- Put strings inside double quotes/single quotes
- Put variable names (without any quotes)
- Separate any sequence of strings and variable names with comma

Eg)

```
print "What's up, ",name,"?"
print fav_color, " is also my favorite color!"
print "I hate the number ",fav_number,"."
print "I don't get what's so great about ",fav_float,"."
```

** Something extra for convenient printing

- formatting decimal points

  {:.2f}.format(*variable name*)

→ format *variable name* (of class `float, double` or `int`) into a string up to the second decimal point

- formatting multiple variables into a string without the continuous use of commas

```
print("pennies:{}, nickles:{}, total: ${:.2f}".format(pennies, nickles, total))
```

→ Literal of `pennies` is formatted into a string, without any additional changes, and put in place of first `{}`
→ Literal of `nickles` is formatted into a string, without any additional changes, and put in place of second `{}`
→ Literal of `total` is formatted into a string until the second decimal point and put in place of `{:.2f}`

E. More methods (if time)

<Strings> suppose `s` is an object of class `str` (in all of these methods, s is unchanged)

- `s.upper()` → returns new string that looks just like s with everything capitalized
- `s.lower()`
- `s.isalpha()` → returns True if the string is comprised solely of alphabets
- `s.split()` → split string into individual elements using whitespace and store in a list
- `s.rstrip(`*something*`)`: returns string with element removed from the right end of string
- `s.lstrip(`*something*`)`: same as above from left end

<LISTS>

- Append and Pop (end of list)

```
In: l.append(5)
In: l
Out: [4,1,'happy', 'munday',5]
In: l.pop()   ## remove element at end of list
In: l
Out: [4,1,'happy', 'munday']
In: l.pop(1) ## you can remove at a specified index
In: l
Out: [4,'happy','munday']
```

- Insert and Remove

```
In: l.insert(2,"sad")   ## insert at index2
In: l
Out: [4,'happy',"sad",'munday']
```

```
In: l.remove("sad") ## For remove, you must give the element,
                    ## not index. (it will remove the first
                    ## occurrence)
In: l
Out: [4,'happy','munday']
```

- Add two lists together

```
In: l.extend(l2)
In: l
Out: [4,'happy','munday',4,3,'sad',1]
```

- Find index of element (This finds the first occurrence)

```
In: l.index(4)
Out: 0
```

- Sort from smallest to largest

```
In: l.sort()
In: l
Out: [1, 3, 4, 4, 'happy', 'munday', 'sad']
```

numbers come before letters in default sorting methods due to ASCII (https://en.wikipedia.org/wiki/ASCII)
to avoid unprecedented complications, only use this when all elements are of the same datatype

- the `in` operator: checks membership in a collection (lists and strings)

```
In: l = [1,2,3,4,5]
In: 5 in l
Out: True
```

<Flip the list around>

```
In: l.reverse()
In: l
Out: ['sad', 'munday', 'happy', 4, 4, 3, 1]
```

** FYI: All of these methods modify the list `l`, instead of creating a new list object. This is because lists are **mutable** objects (We will get to this distinction later in the course)

F. Definition of Big Terms (FYI)

Datatype: A set of values and operations that can be used on them
Class: computer model of datatype (computer representation)

Object: the computer representation of a datatype value

- `x=3` would an object of integer datatype

- Composed of 3 parts
  1) Identity (how it is recognized by humans… here, it's recognized as x!)
  2) Type (datatype)
  3) Set of values (its literal)

Object reference: location in the computer memory of the object

Literal: symbolic representation of a datatype value

- For `x=3`, `x` is the object or, more accurately, an object reference, and `3` is the literal

Identifier: a symbolic representation of a name

- any combination of a sequence of letters, digits, and underscores (**But you cannot begin with a digit**)
- Eg) For `Fav_number = 5`, `Fav_number` would be the identifier)

Keyword: a reserved word in Python that can't be used to name objects, i.e. can't be an identifier

- `import, print, if, else, while, format, math,`
- We will cover the use of these keywords throughout the course of the semester

Expression: a combination of values and operators that evaluates to some data value

- If you can print it or assign it to a variable, it's an expression
- eg. x==5 → print str(x==5) → True/False

Statement: a standalone unit that doesn't return anything

- eg. x=5

** Note there is always a distinction between the computer representation and the concept itself

- e.g. class v.s. data-type
- e,g, object v.s. literal

**What is the difference between a method and function?

Very similar except…

1) the syntax is different

- method: `object_reference.methodname(any parameters)`
  - object reference is an implicit argument of the method
- function: `function_name(object_reference)` or `module_name.function_name(object_reference)`
  - object reference is an explicit argument of the function

2) methods belong to the class (computer representation of a datatype)