Python Fundamentals

Versions (2 vs 3)

- Print function: print variable print(variable)
- Integers: 3/2 = 1, 3/2.=1.5 3/2 =1.5, 3/2.= 1.5
- Looping
- Error Handling
- Details Here: https://sebastianraschka.com/Articles/2014_python_2_3_key_diff.html

Interpretor and Scripting

- Python is used in two ways
- Interpretor:
  - Invoked with python -i
  - Feels similar to a Unix Terminal
  - Use help() or help(*name*) for more information
  - Exit with Ctrl-D or exit()
- Scripting
  - Save Python commands in a file
  - Run with python *file.py*

Printing Text

- The *print* statement displays text on the screen
  - print(“Hello World”)
  - print(“Hello”, “World”)
  - print(3)
  - var = 3; print(var)

Arithmetic

- Standard operators: +, -, *, /
- Exponentials: x ** y
- Modulus (Remainder): x % y
- Normal order of operation applies
  - If you're not sure about order of operation, it never hurts to include parenthesis!
- Be careful when doing math!
  - 2/3 does not equal 2./3 (in Python 2.x)

Variables

- Storage containers for data
- Weakly typed: don't explicitly set a type
  - X = 5 #integer
  - X = 5.0 # float
  - X = “5” # string
  - X = True # Boolean
- Watch out when doing division!
  - float(x)/y to ensure floating point division

Mathematical Libraries

- More complicated routines are stored in a library
- Load a library with the import statement
  - *import math*
    - math.pi
  - *import math as m*
    - m.pi
  - *from math import **
    - pi

$$f(x) = \frac{1}{\sqrt{2\pi}\,s} \exp\left[-\frac{1}{2}\left(\frac{x-m}{s}\right)^2\right]$$

Print Formated Text

- Use '%' as a placeholder in a string
  - print “number: %d” % 3
  - print “string and float: %s : %f” % (“hello”, 3.5)

Format Specifications

- Common format specifications

%s	a string
%d	an integer
%0xd	an integer padded with x leading zeros
%f	decimal notation with six decimals
%e	compact scientific notation, e in the exponent
%E	compact scientific notation, E in the exponent
%g	compact decimal or scientific notation (with e)
%G	compact decimal or scientific notation (with E)
%xz	format z right-adjusted in a field of width x
%-xz	format z left-adjusted in a field of width x
%.yz	format z with y decimals
%x.yz	format z with y decimals in a field of width x
%%	the percentage sign (%) itself

Common Programming Tasks

- Motivational Example:
  - Convert Celsius to Fahrenheit
  - -20 to 40 in 5 degree steps

```
-20   -4.0
-15    5.0
-10   14.0
 -5   23.0
  0   32.0
  5   41.0
 10   50.0
 15   59.0
 20   68.0
 25   77.0
 30   86.0
 35   95.0
 40  104.0
```

Naive Solution

```
C = -20;  F = 9.0/5*C + 32;  print C, F
C = -15;  F = 9.0/5*C + 32;  print C, F
C = -10;  F = 9.0/5*C + 32;  print C, F
C =  -5;  F = 9.0/5*C + 32;  print C, F
C =   0;  F = 9.0/5*C + 32;  print C, F
C =   5;  F = 9.0/5*C + 32;  print C, F
C =  10;  F = 9.0/5*C + 32;  print C, F
C =  15;  F = 9.0/5*C + 32;  print C, F
C =  20;  F = 9.0/5*C + 32;  print C, F
C =  25;  F = 9.0/5*C + 32;  print C, F
C =  30;  F = 9.0/5*C + 32;  print C, F
C =  35;  F = 9.0/5*C + 32;  print C, F
C =  40;  F = 9.0/5*C + 32;  print C, F
```

While Loop

- The ‘While’ Loop
  - Format: while (condition is true) : do something

$$
\begin{aligned}
&C = -20 \\
&\text{while } C \leq 40 \text{ repeat the following:} \\
&\quad F = \tfrac{9}{5}C + 32 \\
&\quad \text{print } C,\ F \\
&\quad \text{set } C \text{ to } C + 5
\end{aligned}
$$

Python While Loop

- Everything to be executed in the loop must be indented.
- You must have a colon (:) after your condition

```
print '------------------'      # table heading
C = -20                        # start value for C
dC = 5                         # increment of C in loop
while C <= 40:                 # loop heading with condition
    F = (9.0/5)*C + 32         # 1st statement inside loop
    print C, F                 # 2nd statement inside loop
    C = C + dC                 # 3rd statement inside loop
print '------------------'      # end of table line (after loop)
```

Boolean Expressions

- So long as the Boolean expression 'C <= 40' evaluates to true, the loop continues

```
C == 40    # C equals 40
C != 40    # C does not equal 40
C >= 40    # C is greater than or equal to 40
C >  40    # C is greater than 40
C <  40    # C is less than 40
```

- We can also using the keyword 'not'
  - i.e. not C == 40
- Boolean expressions can be combined with 'and' or 'or'

```
while x > 0 and y <= 1:
    print x, y
```

Incrementing Variables

- Incrementing variables `C = C + dC` common, that Python has its own operator for it.

```
C += dC   # equivalent to C = C + dC
C -= dC   # equivalent to C = C - dC
C *= dC   # equivalent to C = C*dC
C /= dC   # equivalent to C = C/dC
```

- Note:
  - No C++ or C-- operators like in C/C++

Practicals 1-3

Make a Python script file for each practical

1) Count Down From 10...1

2) Print even numbers between 1 and N

3) Compute the sum

$$s = \sum_{k=1}^{M} \frac{1}{k}$$

Lists

- Variables that store more than one value

```
C = [-20, -15, -10, -5, 0, 5, 10, 15, 20, 25, 30, 35, 40]
```

- The index represents the location of an element in the list
  - Indices start at 0 and end at (length of list)-1
- Access elements by indexing the variable:
  - C[3] → -5
- Not all data is lists must be of the same type

List Insertion Functions

- New values can be put into the list 3 ways
  - Append

```
>>> C = [-10, -5, 0, 5, 10, 15, 20, 25, 30]     # create list
>>> C.append(35)                                # add new element 35 at the end
>>> C                                           # view list C
[-10, -5, 0, 5, 10, 15, 20, 25, 30, 35]
```

  - Addition

```
>>> C = C + [40, 45]                           # extend C at the end
>>> C
[-10, -5, 0, 5, 10, 15, 20, 25, 30, 35, 40, 45]
```

  - Insertion

```
>>> C.insert(0, -15)                           # insert new element -15 as index 0
>>> C
[-15, -10, -5, 0, 5, 10, 15, 20, 25, 30, 35, 40, 45]
```

List Properties

- Length of list found with builtin keyword
  - len(C)
- Find values in list

```
>>> C.index(10)                    # find index for an element (10)
3
```

- Boolean check if value is in list C

```
>>> 10 in C                        # is 10 an element in C?
True
```

- Index wrapping

```
>>> C[-1]                          # view the last list element
45
>>> C[-2]                          # view the next last list element
40
```

Assigning Variables to Lists

- We can pull the values of a list out and assign them to variables
  - var1, var2, var3 = list
  - Number of variables must equal length of list

```
>>> somelist = ['book.tex', 'book.log', 'book.pdf']
>>> texfile, logfile, pdf = somelist
>>> texfile
'book.tex'
>>> logfile
'book.log'
>>> pdf
'book.pdf'
```

For Loops

- Perform the same operation for each element in a list
- Syntax
  - for var in list: do something

```
degrees = [0, 10, 20, 40, 100]
for C in degrees:
    print 'list element:', C
print 'The degrees list has', len(degrees), 'elements'
```

- Very different from traditional C programming

Putting It Together

- Combining both loops and lists

```
Cdegrees = [-20, -15, -10, -5, 0, 5, 10, 15, 20, 25, 30, 35, 40]
Fdegrees = []        # start with empty list
for C in Cdegrees:
    F = (9.0/5)*C + 32
    Fdegrees.append(F)
```

Looping Over Index

- Range function: Returns a list of integers
  - range(N)
    - [0, 1, 2, …, N-1]
  - range(start, stop, step)

```
Cdegrees = range(-20, 45, 5)      # generate C values
Fdegrees = [0.0]*len(Cdegrees)    # list of 0.0 values
for i in range(len(Cdegrees)):
    Fdegrees[i] = (9.0/5)*Cdegrees[i] + 32
```

- Notice [0.0]*len(Cdegrees)

Changing List Elements

- Wrong

```
for c in Cdegrees:
    c += 5
```

  - c is only a copy of the element

- Correct

```
for i in range(len(Cdegrees)):
    Cdegrees[i] += 5
```

Index and Value

- We can get the index and value of a list at the same time using enumerate

```
for i, c in enumerate(Cdegrees):
    Cdegrees[i] = c + 5
```

  - Enumerate returns: index, value

List Comprehension

- Because looping over lists is very common, Python simplifies their manipulation

```
Cdegrees = [-5 + i*0.5 for i in range(n)]
Fdegrees = [(9.0/5)*C + 32 for C in Cdegrees]
C_plus_5 = [C+5 for C in Cdegrees]
```

- Or not!

Practicals 4-6

Make a Python script file for each practical

4) Store odd numbers from 1 to N in a list

5) Calculate cos(x) for x in range(0, 2pi) in 0.1 step increments

6) Compute the mean of numbers in a list

If you have spare time, try using list comprehension!

Nested Lists

- A list can store any data type... including lists

```
Cdegrees = range(-20, 41, 5)   # -20, -15, ..., 35, 40
Fdegrees = [(9.0/5)*C + 32 for C in Cdegrees]

table = [Cdegrees, Fdegrees]
```

- Indexing a list of lists take two indices
  - table[0][2] – Retrieves the 3rd element of Cdegrees

Nested Lists

- If we wanted to access C and F pairs, we can build a new list, looping over values of C and F

```
table = []
for C, F in zip(Cdegrees, Fdegrees):
    table.append([C, F])
```

- Listception Comprehension

```
table = [[C, F] for C, F in zip(Cdegrees, Fdegrees)]
```

- Now table[0] returns [C[0], F[0]]

List Slicing

- Creates a new list from a subset of the original elements of a list
- Slicing a list from index start to end-1:
  - A[start:end]
- Slicing from index I to the end:
  - A[i:]

```
>>> A = [2, 3.5, 8, 10]
>>> A[2:]
[8, 10]

>>> A[1:3]
[3.5, 8]

>>> A[:3]
[2, 3.5, 8]
```

Sublists

- Sublists are copies of the original.
  - Changes to one won't affect the other.

```
>>> l1 = [1, 4, 3]
>>> l2 = l1[:-1]
>>> l2
[1, 4]
>>> l1[0] = 100
>>> l1                  # l1 is modified
[100, 4, 3]
>>> l2                  # l2 is not modified
[1, 4]
```

Nested Loops

- Example: A list of scores for each player in a game
  - [[1, 5, 3], [5, 2, 7, 9, 2], [1, 6]]
  - 3 Players, different number of games

```
for p in range(len(scores)):
    for g in range(len(scores[p])):
        score = scores[p][g]
        print '%4d' % score,
    print
```

```
for player in scores:
    for game in player:
        print '%4d' % game,
    print
```

Functions

- Collection of statements
- Help avoid writing the same code again and again
- Makes code more manageable

Functions

- Defining a function

```
def F(C):
    return (9.0/5)*C + 32
```

- Calling a function

```
a = 10
F1 = F(a)
temp = F(15.5)
print F(a+1)
sum_temp = F(10) + F(20)

Fdegrees = [F(C) for C in Cdegrees]
```

Function Variables

- Variables defined inside a function are 'local' and only exist inside the function.
- Once the function is completed, the variables are removed.
- Variables with the same name are resolved by
    1) Local
    2) Global
    3) Built-in

Function Arguments/Returns

- Multiple arguments

```
def yfunc(t, v0):
    g = 9.81
    return v0*t - 0.5*g*t**2
```

- Multiple return values

```
def yfunc(t, v0):
    g = 9.81
    y = v0*t - 0.5*g*t**2
    dydt = v0 - g*t
    return y, dydt

position, velocity = yfunc(0.6, 3)
```

Keyword Arguments

- Some functions have arguments with default values that don't need to be set
- Sometimes, we want to change these values

```
>>> def somefunc(arg1, arg2, kwarg1=True, kwarg2=0):
>>>     print arg1, arg2, kwarg1, kwarg2

>>> somefunc('Hello', [1,2])
Hello [1, 2] True 0
>>> somefunc('Hello', [1,2], kwarg1='Hi')
Hello [1, 2] Hi 0
>>> somefunc('Hello', [1,2], kwarg2='Hi')
Hello [1, 2] True Hi
>>> somefunc('Hello', [1,2], kwarg2='Hi', kwarg1=6)
Hello [1, 2] 6 Hi
```

Lambda Functions

- Quick 1 line functions

```
def f(x):
    return x**2 + 4

f = lambda x: x**2 + 4
```

- Format

```
def g(arg1, arg2, arg3, ...):
    return expression

g = lambda arg1, arg2, arg3, ...: expression
```

Control Flow and Branching

- Logical idea:
  - If this, then that, else something
- Syntax
  - if condition:
  - elif condition:
  - else:
- elif = else if

If Statements

```
if condition1:
    <block of statements>
elif condition2:
    <block of statements>
elif condition3:
    <block of statements>
else:
    <block of statements>
<next statement>
```

```
def N(x):
    if x < 0:
        return 0.0
    elif 0 <= x < 1:
        return x
    elif 1 <= x < 2:
        return 2 - x
    elif x >= 2:
        return 0.0
```

Practicals 7 & 8

Make a Python script file for each practical

7) Write a function for the Tent Map:

$$\mathrm{f}(x) = \begin{cases} x & 0 < x < 1 \\ 2 - x & 1 < x < 2 \\ 0 & \text{otherwise} \end{cases}$$

8) Write a function that computes the factorial of its input