Lec 7. Introduction to Python

We introduce the syntax, structure and data types in Python. You can test a few statements of Python code in the Python shell.

1. To start a Python shell, open an xterm and type python in the command line.

   ```
   $ python
   ```

   You will see system message and the command prompt >>> as:

   ```
   Python 3.6.4 |Anaconda custom (64-bit)| (default, Jan 16 2018, 18:10:19)
   [GCC 7.2.0] on linux
   Type "help", "copyright", "credits" or "license" for more information.
   >>>
   ```

   Since we will be working with Python 3, make sure the Python version numbers reported on the first line is `python 3.x.y` where the minor revisions `x,y` are not important.

2. To exit the Python shell, type exit().

   ```
   >>> exit()
   ```

Numbers, Variables, Comparisons and Logic

Python has three number types: integer (type: `int`), floating point numbers (type: `float`), complex numbers (type: `complex`)

1. Integers are whole numbers like 1, -11 etc. Integer arithmetic is exact, limitation is restricted only by computer's memory.
2. Floating point numbers are the representation of real numbers like 1.234×10^5 in computer (not always exact). They are stored in binary with certain precision (15-16 decimal places). *A single number containing a dot ('.') is considered a float. Scientific notation is supported by e or E.*
   123400., 1.234e5, 0.01234, 1234e-5
3. A complex number is specified by either adding its real part to an imaginary parts (denoted by `j` suffix) as in *3.0+2.1j, or complex(3.0,2.1).*

Try: int(-3.5), float(10), complex(2,3)

Numbers, Variables, Comparisons and Logic

Airithmetic Operators in Python

+	addition
-	subtraction
*	multiplication
/	floating point division
//	integer division
%	modulus (remainder)
**	exponentiation

Airithmetic Operators Precedence

**	highest precedence
*, /, //, %	
+, -	lowest precedence

Note: Operators of equal precedence are evaluated left to right with the exception of exponentiation (**), which is evaluated right to left.

Try: 6.5/2, 6.5//2, 6.5%2, 8/4/2, 2**2**3, 844487.**5 + 1288439.**5 -1318202.**5

Numbers, Variables, Comparisons and Logic

Methods and attributes of numbers

- There are two built-in mathematical functions: `abs, round`

  Try: abs(-3.14), abs(2+3j), round(5.54), round(9.5), round(4.5)

- Real/imaginary parts and conjugate of complex numbers are access using `object.attribute`

  Try: (2+3j).real, (2+3j).imag, (2+3j).conjugate()

- There are many useful mathematical functions in the `math` module, which can be accessed using `import math` (or `import cmath` for complex functions) in the code.

  Try: math.sqrt(4), math.exp(1), math.log(2), math.log(100,10), math.cos(0), math.atan(1)

- Two useful constants in `math` module: `math.pi, math.e`

  Try: math.sin(math.pi/2), math.log(math.e)

- You can also import `math` module by '`from math import *`' and access its functions directly, but not recommended as there may be name conflicts if you use many modules.

  Try: sin(pi/2), log(e)

For a complete list of the mathematical functions in the `math` module: http://docs.python.org/3/library/math.html

Numbers, Variables, Comparisons and Logic

What is a variable?

When an object, such as a number, is created in Python code, memory is allocated for it. The location of this memory is called its 'address'. It is not convenient to refer to the object by its address, we need a variable name or object identifier.

Some rules for valid variable names

- Variable names are case sensitive.
- Variable names can contain any leeter, underscore, any digits, but cannot start with a digit.
- The following Python *reserved keywords* or *built-in constants* cannot be used for variable names.

  and, assert, break, class, continue, def, del, elif, else, except, False, finally, for, from, global, if, import, in, is, lambda, None, nonlocal, not, or, pass, print, raise, return, True, try, while, yield

- Good practice: make variable names should be meaningful, but not be too long; do not use, I, l, O (similar to 1, 0); use i,j,k for integer counters; use 'max_score' instead of 'MaxScore' (reserved for class names)

Numbers, Variables, Comparisons and Logic

Python Comparison operators

==	equal to
!=	not equal to
>	greater than
<	less than
>=	greater or equal to
<=	less or equal to

- The result of a comparison is a *boolean* object (type `bool`) which is either `True` or `False`.
- Comparison can be modified and combined together with the logic operator keywords `and`, `not`, `or`
- Truth table for `not`, `and` and `or`

P	not P
True	False
False	True

P	Q	P and Q
True	True	True
False	True	False
True	False	False
False	False	False

P	Q	P or Q
True	True	True
False	True	True
True	False	True
False	False	False

Strings in Python

A Python string object (type: `str`) is an ordered, immutable sequence of characters.

- To define a variable containing a string of characters, the string should be enclosed by the either single or double quotes:

```
s1 = "Hello, Sir!  How are you doing?"
s2 = ""                                          (empty string)
```

Try: `print(s1), print(s2)`

- Strings can be concatenated using either + (plus) operator or by placing them next to each other on the same line.

```
s3 = "input"+".dat"   (or s3 = "input" "dat")
```

- Strings can be repeated using * operator.

```
s4 = "abc"*3
s5 = ("aa"+"bb")*2
```

- The built-in function, `str` converts an object passed as its agrument into a string.

```
s6 = str(1e-3)
```

Try: `print(s6), print(4*s6)`

Strings in Python

- Python has no restriction on the length of a line. For readability, 79 characters is recommended. To break up a string over two or more lines, use the line continuation character, '\' or (better) enclose the string in parenthesis.

```
s7 = "This is an example of" \
     "a long string!"


s8 = ("This is another example of"
      "a long string!")
```

- The choice of quotes for the strings allows us to include the quote character inside the string: just define it using the other quote:

```
s9 = "It doesn't make sense!"
```

- If you need to include both quotes in the string, you should use: `\'` for single quotes and `\"` for double quotes:

```
s10 = "She said, \"She isn't worried about Python.\""
```

- Also try `\n` (linefeed), and `\t` (horizontal tab):

```
s11 = "Physics\nChemistry\nMathematics\nBiology"
s12 = "Physics\tChemistry\tMathematics\tBiology"
```

Strings in Python

Indexing and slicing strings:

- *Indexing* a string in Python return a single character at a given position, the first character having index 0 and final character in a string of n characters has index $n-1$. Python raises an `IndexError` if you try to index a string out of its length.

  Set `s = "Embry-Riddle"` and find `s[0]`, `s[6]`, `s[11]`, `s[-1]`, `s[-6]`, `s[12]`

- *Slicing* a string, `s[i:j]`, gives a substring between the characters at two indexes, *including* the first (`i`) and *excluding* the second (`j`). If the first index is omitted, 0 is assumed; if the second is omitted, the string is sliced to its end.

  Try: `s[:5]`, `s[1:6]`, `s[6:]`, `s[:]`

- The optional, third number `k` in a slice `s[i:j:k]` specifies the *stride*. If omitted, default is $k = 1$. Negative value of `k` reverses the string.

  Try: `s[::2]`, `s[1:6:2]`, `s[-1:-6:-1]`

Strings in Python

- Python strings are *immutable* objects, it is not possible to change a string by assignment. Try:

```
s="orange"; s[0]="O"

s1=s[1:]; s2="O"+s1
```

- The built-in function `len` gives the number of characters in a string.

```
len(s)
```

- String objects come with a large number of methods which are accessed using the dot notation. Try:

```
s= "EmbrY_RiddLe aeronutical universiTy"

s.upper(), s.lower(), s.title(), s.title().replace("_", "-").center(72)
```

The `print` function:

- The built-in function `print` in Python takes a list of objects, and optional arguments `end` and `sep` that respectively specify which characters should end the string and which characters should be used to separate the printed objects. Try:

```
print("Solve: ", 2, "x^2 = ",8)
print("Solve: ", 2, "x^2=",8, sep="", end=" !\n")
```

Lec 8. String formatting

- The most basic syntax for string formatting in Python is

```
print("{} plus {} = {}".format(3, 4, "seven"))
print("{1} - {0} = {2}".format(2, 7, 5))
```

- Python string `format` provides a powerful way to format the integers, with specifier 'd'. Try:

```
x=1234; y=-3456; z=-1211
print("x = {0:5d}".format(x))     # with 5 character spaces
print("x = {x:5d}".format(x=x))   # padded with zeros to fill the 5 spaces
print("{0: 5d}"\n{1: 5d}\n{2: 5d}".format(x,y,z))
print("{0:+5d}"\n{1:+5d}\n{2:+5d}".format(x,y,z))
print("{0:-5d}"\n{1:-5d}\n{2:-5d}".format(x,y,z))
```

- Some useful format specifiers for the floating point numbers are: 'f' (fixt-pont notation), 'e' (scientific notation), and 'g' (a general format, uses scientific notation for very small and vvery big numbers). The desired precision (number of decimal places) is specified as '.p' after the minimum field width. Try:

```
x=123400.0056789
print("{0:g}".format(x))
print("{0:10.2e}".format(x))
print("{0:15.10f}".format(x))
```

Next

Classwork 2.

Numbers, Variables, Arithmetic Operations & Strings

Due: 09/20/2018

Structured Programming/Steps of program development

1. Analysis of the problem: numerical methods
2. Divide your method into subprograms: top-down analysis
3. Draw a flow chart: structure plan
4. Translate the flow chart into a computer language and begin with subproblem: bottom-up method
5. Compile your program: compiler checks the program for syntactic errors, translates the instruction into machine language $\rightarrow$ linker errors (if any module or subprogram was not found)
6. Test your program, first check the subprogram solver!
   is it logically correct? does it solve your problem?