Lecture 3: Python Basics

BT 3051 – Data Structures and Algorithms for Biology

Karthik Raman
Department of Biotechnology
Indian Institute of Technology Madras

August 5, 2014

Python Sneak Peek

- Primitive data types
  - Integers (`int`)
  - Floating point numbers (`float`)
  - Strings (`str`)
  - Booleans (`bool`)
- Built-in composite data types
  - Lists (`list`)
  - Tuples (`tuple`)
- Associative arrays (`dict`)
- User-defined data types: objects can be created via class definitions

Python Sneak Peek

- Simple statements
  - Assignment (=)
  - `print`
  - `return`
  - `import`
  - `pass`
- Compound statements
  - `if`, `elif`, `else`
  - Function definitions (`def`)
  - Loops (`for`, `while`, `range`)

Python Sneak Peek: Strings

String methods are very important for biology!

- `find()`
- `count()`
- `index()`
- `join()`
- `replace()`
- `split()`
- `splitlines()`
- `strip()`
- `lower()`
- `upper()`

- Practice, practice, practice!

See: https://docs.python.org/3.1/library/stdtypes.html#string-methods

dir (str)

```
['__add__', '__class__', '__contains__', '__delattr__',
'__dir__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__getitem__', '__getnewargs__', '__gt__',
'__hash__', '__init__', '__iter__', '__le__', '__len__',
'__lt__', '__mod__', '__mul__', '__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__rmod__',
'__rmul__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', 'capitalize', 'casefold', 'center', 'count',
'encode', 'endswith', 'expandtabs', 'find', 'format',
'format_map', 'index', 'isalnum', 'isalpha', 'isdecimal',
'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable',
'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower',
'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex',
'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines',
'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper',
'zfill']
```

Getting help

```
>>> help(str)
Help on class str in module builtins:

class str(object)
 |  str(object='') -> str
 |  str(bytes_or_buffer[, encoding[, errors]]) -> str
 |
 |  Create a new string object from the given object.
 ...
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 ..
 |  upper(...)
 |      S.upper() -> str
 |
 |      Return a copy of S converted to uppercase.
 ...
```

Python has a load of built-in functions

Check out:

```
dir(__builtins__)
```

and more importantly,

```
help(__builtins__)
```

Self-assessment Exercise: Explain an interesting function or class from the module `__builtin__`

Automagic documentation

```
>>> help(str.upper)
Help on method_descriptor:

upper(...)
    S.upper() -> str

    Return a copy of S converted to uppercase.

>>> help('a'.upper)
Help on built-in function upper:

upper(...) method of builtins.str instance
    S.upper() -> str

    Return a copy of S converted to uppercase.

>>> print ('a'.upper.__doc__)
S.upper() -> str

Return a copy of S converted to uppercase.
```

Python Sneak Peek: File Handling

- `open()`
- `read()`
- `readlines()`
- `write()`
- `Advanced: pickle()`

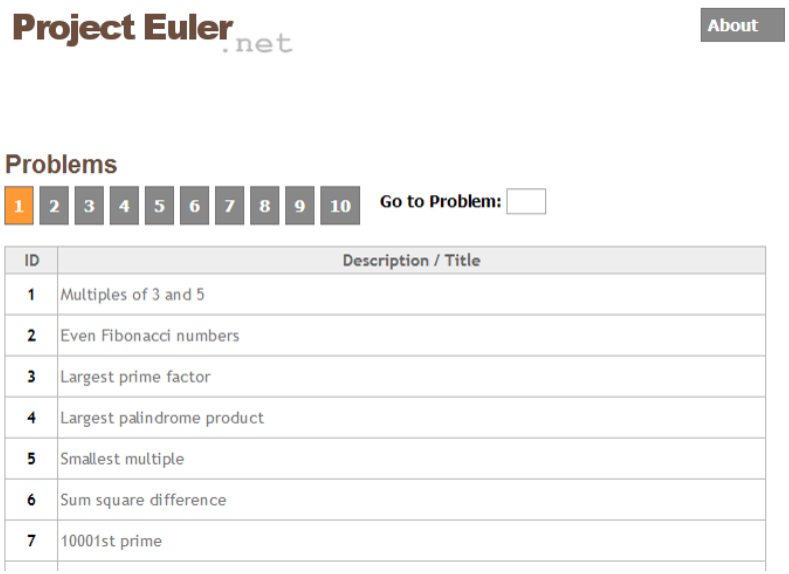
See: https://docs.python.org/2/tutorial/inputoutput.html

Python 3 vs. Python 2

- Since we're starting afresh, let's use Python 3
- However, codeskulptor supports only Python 2
- Python3ifying your code
  - `from __future__ import division`
  - Ensures that `3 / 2 = 1.5` and not `1` (integer division!), even in Python 2
  - `3//2` is integer division in Python 3
  - Does not work on codeskulptor though
- `print` is a function in Python 3
  - `print (1)` instead of `print 1`

Self-assessment Exercise

- Check out the site Project Euler (http://projecteuler.net)
- Has many mathematical problems to solve
- Exercise:
  - Solve a problem on Project Euler
  - Post the solution on piazza (via a codeskulptor URL)
- Outcome:
  - Basic Python use (loops, arithmetic etc.)
  - Familiarity with codeskulptor



Charles Severance on Python/Programming

"Programming is like learning an instrument, it takes practice. Don't expect to be able to play Bach on your first day. There is a steep learning curve when you start, but it gets easier quickly, just keep going. Feeling frustrated is fine, you are learning and it will make sense really soon. The thrill you get when it finally works is indescribable and you will be hooked. Python is an easy language to learn; it's easier than C++ or Objective C. I also think it's easier than Java. Javascript is probably about as easy and some courses choose to teach that. One of the advantages of Python is the number of resources there are to help you learn. Being able to get help when you get stuck is really important at the start. With Python, you can search online and there will usually be an answer. ...I don't think there is any sinister reason why Python is now taught in introductory computer science courses across the world, it's because it is easy for beginners to learn."

Variables

```
phi = 1.618034  # a floating point number
phi = 'golden ratio'  # a string
fib = [1, 1, 2, 3, 5, 8, 13]  # a list in Python
```

- Comments are indicated with '#'
- Triple quotes MUST NOT BE USED for block comments!
- Variables can change in value
- ...and type (unlike C/C++/Java)

Arithmetic Operators

```
>>> 2+4
6
>>> 3*3
9
>>> 3**3
27
>>> 3^3
0
>>> 3/2
1.5
>>> 3//2
1
```

```
>>> 'hi '*10
'hi hi hi hi hi hi hi hi hi hi '
>>> 'Hello, ' + 'World!'
'Hello, World!'
>>> [1,2]*10
[1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1,
                              2, 1, 2, 1, 2]
>>> [1,2]+[3,4,5]
[1, 2, 3, 4, 5]
```

Comparison Operators

```
>>> 2==2
True
>>> 2<3
True
>>> 3>=(1*3)
True
>>> x=1
>>> 0<x<2
True
>>> 0<x<1
False
```

Boolean Operations

```
>>> x=1
>>> y=5
>>> x is 1
True
>>> y is not 5
False
>>> x==1 and y==5
True
```

```
>>> x==1 or y==5
True
>>> x!=1
False
>>> not x==1
False
>>> x is not 1
False
```

- Python uses `and`, `not`, `is` etc. instead of the symbols!
- More readable!
- Python does lazy verification of compound statements (short-circuit): e.g. `if x!=0 and n/x<0.5:`

Control Flow

```
def print_grades(score):
    ''' (int) --> Print a string
    >>> print_grades(70)
    You must work harder!
    >>> print_grades(90)
    S
    >>> print_grades(85)
    A'''
    if 85<score<=100:
        print('S')
    elif 75<score<=85:
        print('A')
    else:
        print('You must work harder!')
```

Loops

while loop

```
def fib_gt_n(nmax):
    fibn=1
    fibn1=1
    n=2
    while True:
        n=n+1
        fib=fibn+fibn1
        fibn1=fibn
        fibn=fib
        if fib>nmax:
            break

    print ('F',n,': ',fib,sep='')
```

```
>>> fib_gt_n(1000)
F17: 1597
>>> fib_gt_n(10000)
F21: 10946
```

Loops
for loop

- Behaviour is markedly different from other languages
- In Python, for **iterates** over elements in an object

```
>>> num=[1,2,3,4,5,6,7]
>>> for i in num:
    print(i**i)

1
4
27
256
3125
46656
823543
```

defining a function

```
def polyval(p, x):
    value = 0
    i = 0
    for coeff in p:
        value += coeff * (x ** i)
        i = i + 1
    return value


>>> polyval([1,1,1,1],4)
85
```

```
def gauss(n):
    return sum(range(n+1))
```

What does the docstring tell us?

```
def print_grades(score):
    ''' (int) --> NoneType
    >>> print_grades(70)
    You must work harder!
    >>> print_grades(90)
    S
    >>> print_grades(85)
    A'''
    if 85<score<=100:
        print('S')
    elif 75<score<=85:
        print('A')
    else:
        print('You must work harder!')
```

Automatic testing

```
>>> import doctest
>>> doctest.testmod(verbose=
                    True)
Trying:
    print_grades(70)
Expecting:
    You must work harder!
ok
Trying:
    print_grades(90)
Expecting:
    S
ok
Trying:
    print_grades(85)
Expecting:
    A
ok
```

```
1 items had no tests:
    __main__
1 items passed all tests:
   3 tests in __main__.
                    print_grades
3 tests in 2 items.
3 passed and 0 failed.
Test passed.
TestResults(failed=0,
                    attempted=3)
```

Did you notice?

- Python uses indentation to group statements/code blocks
- No braces {} like in C/C++
- Each code block, like an `if` or a `for` loop is indented by the same amount
- Python will throw a fit if the indentation is incorrect
- Always use [4] spaces for indentation
- Wrong indentation can also produce wrong code (if you are nesting)

Remember ...

- Practice, practice, practice!
- Document your files/functions
- Comment your code
- Write test cases (make it a habit!)
- How to write good test cases?
- Learn python idioms

Self-assessment Exercise

- Encode Newton-Raphson method for finding the zero of an arbitrary input function
- Input:
  - A Python function and an initial value x_0

```
def newton_raphson(f,x0):
    ...

    return x
```

- Output:
  - Value of x, where $f(x)$ is zero, or an error message if the zero could not be found
- Outcome:
  - Brush up on computational techniques :)
  - Python programming practice

Assignment 0

- Due Tuesday, 12th August @ 17:00
- Work individually
- Write a function to compute the molecular weight of a protein
  - Input: A random protein sequence as a string
  - Output: Molecular weight in Da/kDa
- Trivial assignment
  - Ensures you have Python set up on your computer
  - Tests very basic Python programming
  - Include a `pylint` report with your submission
  - Include a test report, testing your code for some cases
  - Get the discussion going on Piazza!