Lecture 15
Python: strings, file handling

Course: Practical Bioinformatics (BIOL 4220)
Instructor: Michael Landis
Email: michael.landis@wustl.edu

Lecture 15 outline

Last time: if-statements, for-loops,
more with containers

This time: Python (3 of 3)

Python

- strings
- input/output
- filesystem
- system calls

Strings as containers

Substrings may be extracted from strings
using the index operator, *[ ]*

```
>>> x = 'Cookie Monster'
>>> x[0] # return first character
'C'
>>> x[0:6] # return characters 0 through 5 (not 6)
'Cookie'
>>> x[:6] # return up to character before index 6
'Cookie'
>>> x[7:] # return character at index 7 through end
'Monster'
>>> x[0:2] + x[7:9] # concatenate two substrings
'CoMo'
>>> x[ [0,1,2] ] # cannot index string with an index list
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: string indices must be integers
```

String methods

A ***method*** is a function that is associated with a variable, and operates on that variable

All Python strings are equipped with a suite of powerful built-in string-manipulation methods

```
>>> x = 'my shift key is broken'
>>> x.upper()
'MY SHIFT KEY IS BROKEN'
>>> help(x.upper)

Help on built-in function upper:

upper(...) method of builtins.str instance

        S.upper() -> str

        Return a copy of S converted to uppercase.
(END)
```

String methods

Change letters between upper and lowercase

```
>>> x = 'hey, do penguins have DNA?'
>>> x.upper() # all letters to uppercase
'HEY, DO PENGUINS HAVE DNA?'
>>> x.lower() # all letters to lowercase
'hey, do penguins have dna?'
>>> x.title() # 1st letter of each word to uppercase
'Hey, Do Penguins Have Dna?'
>>> x.capitalize() # capitalize 1st letter, rest lowercase
'Hey, do penguins have dna?'
>>> x.swapcase() # change upper to lowercase, and vice versa
'HEY, DO PENGUINS HAVE dna?'
```

String methods

Reformat characters flanking a string

```
>>> x = ' a long pause '
>>> x
' a long pause '
>>> x.strip() # remove all flanking whitespace
'a long pause'
>>> x.lstrip() # remove all whitespace on left
'a long pause '
>>> x.rstrip(' esu') # remove matching chars on right
' a long pa'
>>> x.center(30,'.') # create length-30 string, buffered with .
'....... a long pause .......'
>>> f = '7'
>>> f.zfill(3) # create length-3 string, buffered on left with 0
'007'
```

String methods

Test string properties, return boolean values

```
>>> x = 'Hello' # create string
>>> x.isalpha() # does x only contain alphabetical chars?
True
>>> 'Hello'.isalpha() # call isalpha() method against string value
True
>>> 'hello'.islower() # are all letters in string lowercase?
True
>>> 'HELLO'.isupper() # are all letters in string uppercase?
True
>>> 'h3ll0'.isalnum() # are all characters alphanumeric?
True
>>> '63110'.isdigit() # are all characters numbers?
True
>>> ' \t\n'.isspace() # are all characters whitespace?
True
>>> 'Hello'.startswith('He') # does string start with supplied string?
True
>>> 'Hello'.endswith('lo') # does string end with supplied string?
True
```

String methods

Search for patterns within the string

```
>>> x = 'Shelly sells seashells'
>>> x.count('ell') # how many times does substring appear?
3
>>> x.find('ell') # find index of first occurrence of substring
2
>>> x[2:]
'elly sells seashells'
>>> x.rfind('ell') # find index of last occurrence of substring
18
>>> x[18:]
'ells'
>>> x.find('seashore') # find returns -1 if substring not found
-1
```

String methods

Join and split strings

```
>>> x = 'together forever'
>>> x.split(' ') # tokenize string into list using delimiter
['together', 'forever']
>>> x.replace('er','a') # replace all instances of substring
'togetha foreva'
>>> y = ['b','n','n','j','m']
>>> 'a'.join(y) # use first string as "glue" to concatenate list
'bananajam'
>>> z = 'upstairs\ndownstairs'
>>> z.splitlines(keepends=True) # split string using '\n' delimiter
['upstairs\n', 'downstairs']
```

Chaining methods

Many string methods will return string values upon completion; the returned value can itself call another method! This is sometimes called ***method chaining***.

```
>>> x = 'I am not a crook'
>>> x.upper()
'I AM NOT A CROOK'
>>> x.isupper()
False
>>> x.upper().isupper()
True
>>> 'I am not a crook'[11:].upper().lower().islower()
True
```

Formatting strings

Substitute variables into strings with {*x*} notation

```
>>> mood = 'love'
>>> food = 'donuts'
>>> print('I ' + mood + ' to eat ' + food + '!')
I love to eat donuts!
>>> print(f'I {mood} to eat {food}!')
I love to eat donuts!
>>> print('I {x} to eat {y}!'.format(x=mood, y=food)
I love to eat donuts!
```

Hundreds of ways to format numerical variables

```
>>> '{:06.2f}'.format(3.141592653589793) '003.14'
>>> import datetime
>>> x = datetime.datetime(2020, 11, 7, 12, 39)
>>> '{:%Y-%m-%d %H:%M}'.format(x)
'2020-11-07 12:39'
```

Reading from file

Call *open(filename, ‘r’)* to begin reading a file;
use a for-loop to iterate over each line in the file

```
>>> dirname = '/home/mlandis/'
>>> filename = dirname + 'test.txt'
>>> s = ''
>>> # open the file for reading ('r')
>>> f = open(filename, 'r')
>>> for line in f:
...    s += line + '\n'
...
>>> f.close()
>>> print(s)
upstairs
downstairs
```

Writing to file

Call *open(filename, 'w')* to begin writing to a file; append new content to the file with *f.write(text)*

```
>>> dirname = '/home/mlandis/'
>>> filename = dirname + 'test.txt'
>>> # open the file for writing ('w')
>>> f = open(filename, 'w')
>>> N = 3
>>> for i in range(N):
...   f.write(f'{i+1} of {N}\n')
...
>>> f.close()
>>> quit()
```

code

```
$ cat /home/mlandis/test.txt
1 of 3
2 of 3
3 of 3
```

shell

Example script

```
# filesystem
lab_dirname = '/home/mlandis/labs/lab_14/'
in_filename = lab_dirname + 'input.txt'
out_filename = lab_dirname + 'output.txt’
# store lines in dictionary
x = {}
# read in file
in_file = open(in_filename, 'r')
for i,line in enumerate(in_file):
    # get all fields per row
    fields = line.split(',')
    # ignore header
    if i > 0:
        x[i] = fields

in_file.close()

# write out file
out_file = open(out_filename, 'w')
for k,v in x.items():
    # row elements -> tab-separated string
    row = '\t'.join(v)
    # write each row to file
    out_file.write(row + '\n')

out_file.close()
```

Listing filesystem objects

List all files and directories

```
>>> import os
>>> path = '/home/data_analysis/netflix'
>>> os.listdir(path)
['file.txt', 'docs', 'data']
```

List all files and directories; supports wildcard filters

```
>>> import glob
>>> path = '/home/data_analysis/netflix'
>>> glob.glob(path + "/*.txt")
['file.txt']
```

Function “walks” through part of filsystem and saves files vs. directories

```
>>> import os
>>> path = '/home/data_analysis/netflix'
>>> for root, dirs, files in os.walk(path):
...   for name in files:
...     print(os.path.join(root, name))
...   for name in dirs:
...     print(os.path.join(root, name))
```

Modules

Modules define functions and datatypes that can help solve domain-specific problems

Modules are ***installed*** on a computer then ***imported*** into a Python session to extend the default functionality of the language

```
$ pip install emoji
[ ... installing ... ]

$ python
[ ... initialization text ... ]

>>> import emoji
>>> print(emoji.emojize('Python is :thumbs_up:'))
Python is 👍
```

Anatomy of a module

Modules generally define functions and datatypes, but do not load or process data unless the module is called externally

```python
#!/usr/bin/python
import sys

# add two numbers
def add(a, b):
    return a+b

# multiply two numbers
def mult(a, b):
    return a*b

# behavior if called from command line
if __name__ == "__main__":
    import sys
    a = int(sys.argv[1])
    b = int(sys.argv[2])
    z = add(a, b)
    print(f'{z} = {a} + {b}')
```

babymath.py

Using a module

Ways to access module functions and types

```
>>> import babymath         # import module
>>> babymath.add(2,3)
5
```

```
>>> import babymath as bm      # use shortname for module
>>> bm.add(2,3)
5
```

```
>>> from babymath import add   # import one function from module
>>> add(2,3)
5
```

The __*main*__*()* function will run if the module code is run as a script in Unix

```
$ chmod +x babymath.py
$ ./babymath.py 2 3
5 = 2 + 3
```

Regular expressions

Use the *re* module to use regular expressions.
Use *r’Hello, world!’* syntax to construct string literals for use with regex.

```
>>> import re
>>> s = 'I love to eat donuts’
>>> # find pattern
>>> x = re.findall(pattern=r'.[aeiou].', string=s)
>>> x
['lov', 'to ', 'eat', 'don’]
>>> # replace pattern
>>> y = re.sub(pattern='.([aeiou]).', repl=r'_\1_', string=s)
>>> y
'I _o_e _o__a_ _o_uts’
```

Module contents

List module methods using *dir()*;
Print function definitions with *inspect.getsource(f)*

```
>>> import babymath

>>> # list `babymath` module methods
>>> dir(babymath)
['__builtins__', '__cached__', '__doc__', '__file__’,
 '__loader__', '__name__', '__package__', '__spec__’,
 'add', 'mult', 'sys']

>>> # view function definitions
>>> import inspect
>>> print(inspect.getsource(babymath.add))
def add(a, b):
    return a+b

>>> print(inspect.getsource(babymath.mult))
def mult(a, b):
    return a*b
```

Listing object methods

Use *dir()* with any object to list methods its type

```
>>> # methods for list, [1, 2, 3]
>>> dir([1,2,3])
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__',
'__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
'__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__',
'__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__',
'__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend',
'index', 'insert', 'pop', 'remove', 'reverse', 'sort']

>>> # methods for string, 'a'
>>> dir('a')
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__',
'__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__',
'__getnewargs__', '__gt__', '__hash__', '__init__', '__init_subclass__',
'__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', 'isascii', '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']
```

System calls

Multiple ways to dispatch commands to operating system and retrieve output

```
>>> import os
>>> cmd = 'ls -lart'
>>> out = os.popen(cmd).readlines()
>>> print(''.join(out))
total 12
drwxrwxr-x 10 mlandis mlandis 4096 Nov 10 10:17 ..
-rwxrwxr-x  1 mlandis mlandis  305 Nov 10 12:54 babymath.py
drwxrwxr-x  2 mlandis mlandis 4096 Nov 10 13:38 .
```

using *os.popen()*

```
>>> import subprocess
>>> cmd = 'ls -lart'
>>> p = subprocess.Popen(cmd, shell=True, stdout=subprocess.PIPE)
>>> out = p.stdout.readlines()
>>> for i,o in enumerate(out):
...   out[i] = o.decode('UTF-8')
...
>>> print( ''.join(out) )
total 12
drwxrwxr-x 10 mlandis mlandis 4096 Nov 10 10:17 ..
-rwxrwxr-x  1 mlandis mlandis  305 Nov 10 12:54 babymath.py
drwxrwxr-x  2 mlandis mlandis 4096 Nov 10 13:38 .
```

using *subprocess.Popen()*

Overview for Lab 15