Lab 1: Introduction to Python

Data Structures and Algorithms for CL III
Oct 25 2019

Installation and setup

Installing the latest stable Python

- For this course we will use the latest stable version of Python 3
- On Linux, we can install this at the terminal with
  - sudo apt-get update
  - sudo apt-get install python3
- In general, on any system, go to
  - https://www.python.org/downloads/
- Once installed, you can run Python code directly in the Read-Eval-Print-Loop (REPL) in the terminal with the command:
  - python3
- Or from a file with:
  - python3 path/to/my/script.py
- Or in editors and IDEs using the graphical user interface

Installing a Python editor

- We suggest installing Visual Studio Code, which is a programming text editor
  - Features such as syntax highlighting, syntax checking and debugging for many languages including Python
  - Relatively low overhead and multiplatform (Linux, Mac OS, Windows)
- You are welcome to use whatever you find most comfortable, from IDEs like Pycharm to vim or emacs or (if you must, but please don’t) Notepad
- To install VS Code on your system, go to https://code.visualstudio.com/docs/setup/setup-overview and follow the instructions for your platform
- On Debian Linux systems like Ubuntu, we suggest downloading the .deb install package and running:
  - sudo apt install ./<file>.deb

Getting VS Code in Python-shape

- Once you have VS Code running, install the following extension for Python https://marketplace.visualstudio.com/items?itemName=ms-python.python
- Next, verify your Python installation (on Linux) with
  - python3 --version
- Finally, select the correct Python interpreter within VS Code through the Command Palette (open this with Ctrl+Shift+P)
  - Start typing Python: Select Interpreter and then select the command
  - Choose your stable Python 3 version you just installed

Installing packages for Python

- Just as we import Java libraries, we will frequently need third party packages to get work done in Python
- In general, you can use a program called pip to install packages
  - For example, to install the data science package numPy:
    - python3 -m pip install numpy
- However, Python packages are universally installed on the system unless otherwise handled
  - This can cause version conflicts if different Python programs need different package versions to run on the same system
- Therefore we encourage you to create a “virtual environment” in the directory with your Python files, by the following commands:
  - python3 -m venv .venv
  - source .venv/bin/activate

Basics of Python

Key differences from Java: language features

- Interpreted vs. compiled
  - Java is a compiled language, which means the program semantics are checked by the compiler before running
  - Python is interpreted, which means it is run line by line (permitting code changes while running, though not advisable)
- Static vs. dynamic typing
  - In Java, we statically define types of variables: that is, int blah = 5; or string foo = “words”
    - We cannot arbitrarily later assign a string to an int
    - The compiler will know before runtime if you attempt to compare the two with > for example
  - In Python, types are dynamically assigned by the interpreter: we simply write x = 5 or x = “words” with no prefix
    - A variable x can change types by simply assigning a different value
    - Type incompatibility can frequently only be found at runtime

Key differences from Java: practical preliminaries

- Statements
  - In Java, a statement ends in a semicolon, which lets you place them with arbitrary whitespace in between
  - In Python, whitespace is very important and defines both statement endings (newlines) and scope (indentation of lines)
- Printing strings to console
  - In Java, you would write System.out.print() or .println()
  - In Python, print() adds a newline to the end of each statement by default
    - To override this behavior, you can use print("Hello", end = "") for example
- Comments
  - In Java, you use //normalcomment for single lines and /* this is a block comment */ for block comments
  - In Python, it is #normalcomment for single lines and ''' this is a block comment '''

Key differences from Java: practical preliminaries

- Printing values
    - In Python, string literals are encased in either single quotes or double quotes
        - No style preference but be consistent
    - You can print concatenated strings or string literals with +, not unlike Java
        - print(“Hello ”+”world”)
    - To print other types correctly, use a comma
        - print(“y=”, 5)
- To control whitespace in output, pass sep paramater to the print function
    - By default in Python, just as print() adds a newline, it also adds a space between values concatenated with a comma, but we can override this
        - print(“y= “, 5, sep=””)
- Console input
    - To get console input in Java is a wordy multi-line affair
    - In Python, one can just make a call to built-in function input() to get a value from the console
        - y = input()

Key differences from Java: dealing with strings

- Printing values
  - In Python, string literals are encased in either single quotes or double quotes
    - No style preference but be consistent
  - You can print concatenated strings or string literals with +, not unlike Java
    - print(“Hello ”+”world”)
  - To print other types correctly, use a comma
    - print(“y=”, 5)
- To control whitespace in output, pass sep paramater to the print function
  - By default in Python, just as print() adds a newline, it also adds a space between values concatenated with a comma, but we can override this
    - print(“y= “, 5, sep=””)
- Console input
  - To get console input in Java is a wordy multi-line affair
  - In Python, one can just make a call to built-in function input() to get a value from the console
    - y = input()

Arithmetic operations

- Addition, substraction, multiplication, division
  - The first three are all the same
  - But division acts differently
  - In Java, the / is treated as integer division by default, so
    - 1 / 4 = 0
  - In Python, this is not the case
    - 1 / 4 = 0.25
  - Double // is used for integer division in Python
- Booleans are a subclass of int, meaning they are comparable
  - 1 == True, 0 == False
- Functions like min, max, etc. are built-in and do not require invoking a namespace like Math.min() in Java

Sequences

- In Java, substrings were a somewhat messy affair -- in Python there is much more elegant syntax
- A sequence in Python has a sophisticated set of associated methods for access and manipulation
  - A string is also an (immutable) sequence (of characters)
- There are many kinds of sequences in Python, but for example, to initialize an empty list, just write x = []
  - Or give it some initial values with x = [‘a’, ’b’, ‘c’, 4, ‘d’]
- We can take subsequences much more easily with the “slicing” syntax
  - x[n : k] returns a subsequence starting from index n and ending at index k (exclusive)
  - Leave a blank for n to start from the beginning or a blank for k to go to the end
  - So x[: 3] takes the first three elements, and x[3 :] starts from position 3 and goes to the end
  - Negative indices count from the end instead of the beginning -- for example, x[-3 :] takes the last three elements

Lists, tuples, sets, ranges and strings

- Lists, tuples, strings, ranges and sets are all accessable with sequence syntax
- Lists (analogous to ArrayList objects in Java)
  - Initialized within [] brackets
  - Mutable
  - Any objects/values can be elements
- Tuples
  - Array-like, immutable, initialized like x = (z, y)
  - Any objects/values can be elements
- Sets
  - Unordered, mutable, initialized like x = {‘a’, ‘b’}
  - Hashable objects can be elements
- Strings
  - Immutable sequence of characters
  - Initialized x = “words”
- Ranges
  - Immutable, like strings for numbers, initialized like y = range(10)

Initial assignment 0.1 : Language guessing

- Go here to find the first assignment:
  - https://github.com/dsacl3-2019/a0.1
- We will discuss solutions on Nov 8th -- recall there is no class on Nov 1st!

Arithmetics

```
>>> 1 == 1
True
>>> 1 == 1.0
True
>>> 1 == int("1")
True
>>> 1 == float("1")
True
>>> # bool is a subclass of int
... 1 == True
True
>>> 0 == False
True
```

Arithmetics

```
>>> (1 + 2) * 3
9
>>> 4 ** 2
16
>>> 4 ** 0.5
2.0
>>> # float division is the default
... 5 / 3
1.6666666666666667
>>> 5 // 3
1
```

Sequences

```
>>> mylist = ['a', 'b', 3, 'd', 'e']
>>> mylist[0] # indexing
'a'
>>> mylist[4] == mylist[-1] == 'e'
True
>>> mylist[:2] # slicing
['a', 'b']
>>> mylist[-2:]
['d', 'e']
>>> mylist[1:4]
['b', 3, 'd']
>>> mylist[::2] # steps
['a', 3, 'e']
>>> mylist[::-1]
['e', 'd', 3, 'b', 'a']
```

Sequences

```
>>> mylist = ['a', 'b', 3, 'd', 'e']
>>> mylist + ['f', 'g']
['a', 'b', 3, 'd', 'e', 'f', 'g']
>>> mylist * 2
['a', 'b', 3, 'd', 'e', 'a', 'b', 3, 'd', 'e']
>>> 'b' in mylist
True
>>> 'c' not in mylist
True
>>> otherlist = [1, 1, 3, 3, 2, 4, 1]
>>> otherlist.count(3)
2
>>> otherlist.index(1) # first occurrence
0
>>> otherlist.index(1, 2)  # first occurrence at/after index 2
6
>>> len(otherlist)
7
```

Sequences

```
>>> otherlist = [1, 1, 3, 3, 2, 4, 1]
>>> # for sequences of numbers
... sum(otherlist)
15
>>> min(otherlist)
1
>>> max(otherlist)
4
```

```
>>> nestedlist = [['a', 'b', 'c'], [1, 2, 3], 4]
>>> len(nestedlist)
3
>>> nestedlist[0]
['a', 'b', 'c']
>>> nestedlist[0][1]
'b'
```

Sequences

```
>>> l = ['a', 'b', 'c', 'd', 'e']
>>> for idx, elem in enumerate(l):
...     '{} at index {}'.format(elem, idx)
...
'a at index 0'
'b at index 1'
'c at index 2'
'd at index 3'
'e at index 4'
>>> {elem: idx for idx, elem in enumerate(l)}
{'b': 1, 'e': 4, 'c': 2, 'd': 3, 'a': 0}
>>> l2 = ['v', 'w', 'x', 'y', 'z']
>>> {elem_l: elem_l2 for elem_l, elem_l2 in zip(l, l2)}
{'b': 'w', 'e': 'z', 'c': 'x', 'd': 'y', 'a': 'v'}
```

List

List: a **mutable** array

```
>>> mylist = ['a', 'b', 3, 'd', 'e']
>>> mylist[2] = 'c'
>>> mylist
['a', 'b', 'c', 'd', 'e']
>>> mylist += ['f', 'g']
>>> mylist.extend(['h', 'i'])
>>> mylist.append('j')
>>> mylist
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
>>> mylist.remove('c')
>>> mylist
['a', 'b', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
>>> mylist.insert(2, 'c')
>>> mylist
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
```

List

List: a **mutable** array

```
>>> mylist = ['a', 'b', 3, 'd', 'e']
>>> elem = mylist.pop(2)
>>> "elem: {}, mylist: {}".format(elem, mylist)
"elem: 3, mylist: ['a', 'b', 'd', 'e']"
>>> elem = mylist.pop()
>>> "elem: {}, mylist: {}".format(elem, mylist)
"elem: e, mylist: ['a', 'b', 'd']"
>>> mylist.reverse()
>>> mylist
['d', 'b', 'a']
>>> mylist.sort()
>>> mylist
['a', 'b', 'd']
>>> mylist.clear()
>>> mylist
[]
```

Tuple

Tuple: an **immutable** array

- the general sequence operations also work for tuples

```
>>> mytuple = (1, 2, 3, 4)
>>> mytuple[-2:]
(3, 4)
>>> mytuple + (5, 6) # returns a new tuple
(1, 2, 3, 4, 5, 6)
>>> len(mytuple)
4
>>> 2 in mytuple
True
>>> # immutable!
... mytuple[0] = 2
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
TypeError: 'tuple' object does not support item assignment
```

Range

Range: an **immutable** sequence of numbers

- the general sequence operations also work for ranges
- takes a small amount of memory that does not increase with the length of the sequence

```
>>> myrange = range(10) # 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
>>> myrange[8]
8
>>> myrange[:-3]
range(0, 7)
>>> 7 in myrange
True
>>> tuple(myrange)
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
>>> list(range(1, 9)) # creating ranges is similar to slicing
[1, 2, 3, 4, 5, 6, 7, 8]
>>> list(range(1, 10, 2))
[1, 3, 5, 7, 9]
```

String

```
>>> 'g' in s
True
>>> 'ling' in s # subsequence testing
True
>>> 'computational {}!'.format(s)
'computational linguistics!'
>>> t = "   I'm surrounded by whitespace  \n"
>>> t.strip()
"I'm surrounded by whitespace"
>>> t.split() # default delimiter: whitespace
["I'm", 'surrounded', 'by', 'whitespace']
```

Set

Set: a **mutable**, unordered collection of hashable objects

- mutable containers (sets, dictionaries) are **not** hashable

```
>>> myset = {3, 2, 'a', 'b'}
>>> # collection operations
... 5 in myset
False
>>> len(myset)
4
>>> # set-theoretic operations
... mysubset = {2, 3}
>>> mysubset.issubset(myset)
True
>>> myset.intersection(mysubset)
{2, 3}
```

Frozenset

Frozenset: an **immutable** set

```
>>> myset = {3, 2, 'a', 'b'}
>>> myfrozenset = frozenset(['a', 'b', 2, 3])
>>> myset == myfrozenset # compares members
True
```

Dictionary

Dictionary: a **mutable** mapping from hashable objects to arbitrary objects

```
>>> mydict = {'a': 1, 'b': 2, 'c': 3}
>>> mydict['a']
1
>>> mydict.items()
dict_items([('b', 2), ('c', 3), ('a', 1)])
>>> mydict.keys()
dict_keys(['b', 'c', 'a'])
>>> mydict.values()
dict_values([2, 3, 1])
>>> # can be used as a switch
... switchdict = {'sum': sum, 'len': len, 'min': min}
>>> switchdict['len'](mydict)
3
```

Comparisons

```
>>> 1 < 2 < 3
True
>>> True or 1/0 # lazy! no ZeroDivisionError
True
>>> not False and not None and not 0 \
... and not '' and not () and not [] and not {}
True
>>> l = [1, 2, 3]
>>> m = [1, 2, 3]
>>> # equality
... l == m
True
>>> # identity
>>> l is m
False
>>> l is not m
True
```

`if`, `elif`, `else`

`not`, `and`, `or`

Loops and List Comprehension

```
>>> squares = []
>>> for n in range(10):
...     squares += [n * n]
...
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> n # loops don't create a lexical scope!
9
>>>
>>> # as a one-liner (list comprehension)
... squares = [n * n for n in range(10)]
>>>
>>> some_squares = [x for x in squares if x % 3 == 0]
>>> some_squares
[0, 9, 36, 81]
```

Loops

```
>>> my_set = {'a', 'b', 'c', 'd', 'e'}
>>> while True:
...     elem = my_set.pop() # removes a random member
...     if elem == 'c':
...         break
...     print(elem)
...
a
>>> my_set
{'e', 'b', 'd'}
```

Iterators

```
>>> my_set = {'a', 'b', 'c', 'd', 'e'}
>>> it = iter(my_set)
>>> while True:
...     try:
...         print(next(it))
...     except StopIteration:
...         break
...
a
c
e
b
d
```

Functions

```
def count_vowels(s, vowels=('a', 'e', 'i', 'o', 'u')):
    counts = {}
    for vowel in vowels:
        counts[vowel] = s.count(vowel)
    return counts
    # or all in one line:
    # return {vowel: s.count(vowel) for vowel in vowels}

print(count_vowels('linguistics'))
# {'u': 1, 'a': 0, 'o': 0, 'e': 0, 'i': 3}
```

built-in functions:
`https://docs.python.org/3/library/functions.html`

Functions

```python
def my_function(arg1, arg2='default value', *args, **kwargs):
    print('obligatory:', arg1)
    print('optional:', arg2)
    print('optional (positional):', args)
    print('optional (with keyword):', kwargs)

my_function(0, 1, 2, 3, 4, five=5, six=6)
# obligatory: 0
# optional: 1
# optional (positional): (2, 3, 4)
# optional (with keyword): {'five': 5, 'six': 6}
my_function(0)
# obligatory: 0
# optional: default value
# optional (positional): ()
# optional (with keyword): {}
```

Functions

```python
def my_function(arg1, arg2='default value', *args, **kwargs):
    print('obligatory:', arg1)
    print('optional:', arg2)
    print('optional (positional):', args)
    print('optional (with keyword):', kwargs)

my_list = [3, 4, 5]
my_function(0, *my_list) # unpacks the list
# obligatory: 0
# optional: 3
# optional (positional): (4, 5)
# optional (with keyword): {}
```

Functions

Avoid using mutable default arguments:

```
>>> def add(x, l=[]):
...     l.append(x)
...     return l
...
>>> add(1)
[1]
>>> add(2)
[1, 2]
>>> def add(x, l=None):
...     if l is None:
...         l = []
...     l.append(x)
...     return l
...
>>> add(1); add(2)
[1]
[2]
```

Functions

Functions can return multiple objects:

```
>>> def square_and_cube(x):
...     return x**2, x**3
...
>>> n = 3
>>> s, c = square_and_cube(n)
>>> 'n: {}, s: {}, c: {}'.format(n, s, c)
'n: 3, s: 9, c: 27'
```

File I/O

```
f = open('file.txt', 'w', encoding='utf8')
f.write('Hi!')
f.close()

# Using `with` closes the input stream automatically,
# even if an exception is raised!
with open('file.txt', 'w', encoding='utf8') as f:
    f.write('Hi!')

with open('file2.txt', 'r', encoding='utf8') as f:
    for line in f:
        print(line + '!')

with open('file2') as f:
    list_of_lines = f.readlines()
```

https://docs.python.org/3/tutorial/inputoutput.html