Python 3 Basics

Eddie Guo

September 2019

1 Introduction to Python Basics

1.1 Topics Covered

(i) Interpreted vs compiled code

(ii) Programming style: comments, PEP8

(iii) Simple input/output

(iv) Values and variables

(v) Introduction to built-in data types & how to use them

2 Intro

2.1 Programming = Data + Algorithms

Data strucs + pseudo-code algorithms → programming lang → compiled/interpreted into machine code

- Why are computers dumb?
  - They take instructions literally.
- Why are computers good?
  - B/c they do things over and over rly fast.
- Program = set of instrucs given to computer.
- Computers understand machine lang (1s & 0s; i.e., CPU **only** understands machine lang).
- Unsolvable problems are not computable
- Programs CANNOT exist w/out algorithms

Interpreter (ex: Python 3)

- Interpreter translates program line-by-line until it meets 1st error/end of program.
- Code interpreted every time you run your program.

Compiler (ex: C++)

- Translates entire program into machine code efficiently (execution usually faster).
- Code only compiled when new executable req.

2.2 Python Program Style Notes

- Always include header.
- In header, always include what your program does.
- Comments improve code readability & maintainability.
  - Should explain approach of code (the 'why,' not line-by-line description).
- To check style of helloworld.py, type `style helloworld.py` in terminal.

```
# ==============================
#   Name: Eddie Guo
#   ID: 1576381
#   CMPUT 274, Fall 2019
#
#   Exercise 1: Hello World.
#   Description here
# ==============================
```

2.3 More Python Notes

- Python is dynamically typed.
  - i.e., don't have to explicitly declare variable along w/ type (C++ is diff).
- Any var not assoc w/ var is periodically deleted from mem by Python's garbage collector.

3 Python Variable Names

- Python keywords can't be used as var names (ex: and, as, in, class).
- Variable names also called identifier.
- Underscore ex: `hello_world`
- Lower camel case ex: `dogsTasteGoodLol`
- Upper camel case ex: `MyNameIsJeeeeffffff`
- According to PEP 8, use underscore for multi-word identifiers.

and	del	from	not	while
as	elif	global	or	with
assert	else	if	pass	yield
break	except	import	print	
class	exec	in	raise	
continue	finally	is	return	
def	for	lambda	try	

Figure 1: Common Python 3 keywords

4 Built-In Types & Methods

Immutable means var can't change in place; a new obj is created for each operation. You must assign a variable to ref & store the new obj. Else, garbage collector will remove.

4.1 Built-In Types: `int, float, complex`

- `int, float, complex` are immutable
- `5//2` returns `2` (floored division)
- `-7//3` returns `-3` (floored division)
- `5%2` returns `1` (modulo operator)

4.2 Convert Type

- Can convert type of one type to another (ex: `list()`, `set()`).
- Can't mix types when performing operation (ex: `'CMPUT'+12.0` → `'CMPUT'+str(12.0)`).

4.3 Built-in Types: `bool`

- Boolean is immutable
- Remember the truth tables
- For 'and', only if both operands `True`, then True
  - if 1st item is `False`, don't eval 2nd part
- For 'or', only if both operands `False`, then false
  - If 1st item is `True`, don't eval 2nd part.
- Strings are indexed starting from 0
  - If `myVar='CMPUT'`, then `myVar[2]='P'`
- `replace(old, new, max)` method
- Rem that `strip()!=split(char)`

4.4 String Method: `format()`

```
>>> print('my number is {:15}!'.format(1))
'my number is 1!'

"""
{:<} - left-justified in field width
{:^} - centerized in field width
{:>} - right-justified in field width
{:015} - pad w/ 0s for field width of 15
"""

>>> print('my number is {:15}!'.format(1))
>>> print('my number is {:015}!'.format(1))
my number is               1!
my number is 000000000000001!

"""
{15.2f} - 2 digits after decimal pnt
{0} - mapping 1st element in str to 1st argument in format()
"""

name = 'Fred'; amount = 5.43
>>> print('The person {0:^015} has {1:>07.2f} dollars'.format(name, amount)
The person 00000Fred000000 has 00005.4 dollars
""" NOTES:
    - 1st arg in format centerized w/ Fred in middle, field width = 15, empty spaces filled by 0s, 5 0s on left, 6 0s on right
    - 2nd arg in format right-aligned, padded w/ 0s, width = 7, 2 decimal places
    - format() may come in handy for making tables
"""
```

4.5 Built-In Type: `list`

- List is seq of values of *any* type & is mutable.
- Operators `+`, `*` concatenate list; `:` slices lists
  - `[1,2,3]+[4,5,6]` returns `[1, 2, 3, 4, 5, 6]`
  - `[1,2,3]*3` returns `[1, 2, 3, 1, 2, 3, 1, 2, 3]`
- `k=[1,2,3,4,5,6]`
  - `k[2:3]` returns `[3, 4]`
  - `k[2:]` returns `[3, 4, 5, 6]`
  - `k[:4]` returns `[1, 2, 3, 4]`
- Membership operator `in` asks whether item is in list.
  - `3 in [1,2,3,4,5,6]` returns `True`
  - `len([1,2,3,4,5,6])` returns `6`

4.6 List Methods

- `append()` adds item at end of list
- `insert(i, item)` inserts item at i^{th} pos of list
- `extend(iterable)` appends all items in iterable
- `pop()` removes & returns last item in list
  - `pop(i)` removes & returns i^{th} element in list
- `del list[i]` removes i^{th} element in list
  - `del k[2]` deletes item at index 2 from k
- `remove(item)` removes 1^{st} occurrence of item
- `sort()` modifies list to be sorted
- `reverse()` reverses order of items in list
- `count(item)` returns number of occurrences of item in list
- `index(item)` returns index at 1^{st} occurrence of item

```
>>> list('CMPUT')
['C', 'M', 'P', 'U', 'T']
```

```
>>> '1,2,3,,5'.split(',')
['1', '2', '3', '', '5']
>>>'the cat sat on the mat'.split()
['the', 'cat', 'sat', 'on', 'the', 'mat']
>>>'the,cat,sat,on,the,mat'.split(',',3)
['the', 'cat', 'sat', 'on,the,mat']

>>> ' '.join(['1','2','3','4','5'])
'1 2 3 4 5'
>>> ''.join(['1','2','3','4','5'])
'12345'
>>>  '**'.join(['1','2','3','4','5'])
'1**2**3**4**5'

>>> x = [1,2,3,4,5]
>>> x.reverse()
>>> x
[5, 4, 3, 2, 1]

# Note that del x[len(x)-1] removes the same value as x.pop()
# However, del x[len(x)-1] != x.pop()
```

4.7 Built-In Types: `tuple, set`

- Tuple is immutable list
  - ex: (2, True, 'cat', [1,2,3], 3.5)
- Can't change content of tuple, but can change mutable objs in tuple
  - i.e., can change content of set in tuple
- Set is unordered collection of unique immutable objs, but set itself is mutable
  - ex: 2, True, 'cat', 3.5
  - **CANNOT include lists in sets**
- Sets do not support indexing
- Sets support methods like:
  - `union` or `|`
  - `intersection` or `&`
  - `issubset` or `<=`
  - `difference` or `-`
  - `add(item), remove(item), clear(), pop()`

```
>>> k = (2, True, 'cat', [1,2,3])
>>> print(k[2])
>>> k[2].append(4)
>>> print(k[2])
[1, 2, 3]
[1, 2, 3, 4]
```

5 Aliasing

- `x=y` does NOT make copy of y
- `x=y` makes x ref same obj that y refs CURRENTLY
- Use aliasing ONLY as 2nd name for MUTABLE obj.
  - Aliasing for immutable objs is tricky.
- Aliasing can cause problems:

```
>>> first_var = 'CMPUT'
>>> second_var = first_var
>>> first_var = first_var + '275'
>>> print(first_var)
>>> print(second_var)
CMPUT 275
CMPUT
```