Python Cheat Sheet

JUST THE BASICS
CREATED BY: Arianne Colton and Sean Chen

GENERAL
- Python is case sensitive
- Python index starts from 0
- Python uses whitespace (tabs or spaces) to indent code instead of using braces.

HELP
- Help Home Page: help()
- Function Help: help(str.replace)
- Module Help: help(module)

MODULE (AKA LIBRARY)
Python module is simply a ‘.py’ file
- List Module Contents: dir(module)
- Load Module: import module
- Call Function from Module: module.func()

* import statement creates a new namespace and executes all the statements in the associated .py file within that namespace. If you want to load the module’s content into current namespace, use ‘from module import *’.

SCALAR TYPES

Check data type: type(variable)

SIX COMMONLY USED DATA TYPES
1. int/long* - Large integer automatically converts to long
2. float* - 64 bits, there is no ‘double’ type
3. bool* - True of False
4. str* - ASCII valued in Python 2.x and Unicode in Python 3
   - String can be in single/double/triple quotes
   - String is a sequence of characters, thus can be treated like other sequences
   - Special character can be done via \\ or preface with:
     str = r‘this\r\n\tf
   - String formatting can be done in a number of ways
     temp = ‘4.2f %a hana $4d’;
     str = template % (4.28, ‘Hola’, 2)

DATA STRUCTURES

Note: All non-Get function call i.e. list1.sort() examples below are in-place (without creating a new object) operations unless noted otherwise.

SCALAR TYPES

* str(), bool(), int() and float() are also explicit type cast functions.

5. NoneType(None) - Python ‘null’ value (ONLY one instance of None object exists)
   - None is not a reserved keyword but rather a unique instance of 'NoneType'
   - None is common default value for optional function arguments:
     def func(a, b, c = None):

   - Common usage of None:
     if variable is None:

6. datetime - built-in python ‘datetime’ module
   - ‘datetime’ combines information stored in ‘date’ and ‘time’
   - ‘datetime’ provides ‘date’, ‘date’, ‘time’ types.
   - ‘datetime’ combines information stored in ‘date’ and ‘time’

Example:

```python
from datetime import datetime

dt1 = datetime.

# print 'year/month/day'
print(dt1.strftime('%Y/%m/%d'))
```

Note: Most objects in Python are mutable except for ‘strings’ and ‘tuples’

LIST

One dimensional, variable length, mutable (i.e. contents can be modified) sequence of Python objects of ANY type.

** Create List
list1 = [1, ‘a’, 3] or
list1 = list(tup1)

** Create Nested List
list1 + list2 or
list1.extend(list2)

** Append to End of List
list1.append(b)

** Insert to Specific Position
list1.insert(posIdx, b)

** Inverse of Insert
valueAtIdx = list1.pop(posIdx)

** Remove First Value from List
list1.remove(a)

** Check Membership
3 in list1 -> True

** Sort List
list1.sort()

** Sort with User-Supplied Function
list1.sort(key = len)

Note: `list1.sort()` is O(n log n) for the sorting algorithm used).

** List concatenation using ‘+‘ is expensive since a new list must be created and objects copied over. Thus, extend() is preferable.

** Insert is computationally expensive compared with append.

** Checking that a list contains a value is lot slower than dicts and sets as Python makes a linear scan where others (based on hash tables) in constant time.

Built-in ‘bisect module’:
- Implements binary search and insertion into a sorted list
- ‘bisect.bisect’ finds the location, where ‘bisect.insort’ actually inserts into that location.

Valid dict key types:
- Keys have to be immutable like scalar types (int, float, string) or tuples (all the objects in the tuple need to be immutable too)
- The technical term here is ‘hashability’, check whether an object is hashable with the ‘hash()’ function, hash((1,2)) - this would fail.

** 'KeyError' exception if the key does not exist.

** ‘get()’ by default (aka no ‘defaultValue’) will return ‘None’ if the key does not exist.

** Returns the lists of keys and values in the same order. However, the order is not any particular order, aka it is most likely not sorted.

DICT (HASH MAP)

Create Dict
[‘key1‘: ‘value1‘, ‘key2‘: 2]

Create Dict from Sequence
dict(zip(keyList, valueList))

Get/Set/Insert Element
- dict[‘key1‘] = ‘value1‘
- dict[‘key1‘] = newValue
- dict.get(‘key1‘, defaultValue)

Check if Key Exists ‘key1‘ in dict1

Delete Element
- del dict1[‘key1‘]

Get Key List
- dict1.keys()***

Get Value List
- dict1.values()***

Update Values
- dict1.update(dict2)
- dict1.values() = list1

Note: ‘*’ KeyError exception if the key does not exist.

** ‘get()’ by default (aka no ‘defaultValue’) will return ‘None’ if the key does not exist.

*** Returns the lists of keys and values in the same order. However, the order is not any particular order, aka it is most likely not sorted.

SET

- A set is an unordered collection of UNIQUE elements.
- You can think of them like dicts but keys only.

Create Set
set([3, 6, 3]) or
set([3, 6, 3])

Test Subset
set1.issubset(set2)

Test Superset
set1.issuperset(set2)

Test sets have same content
set1 == set2

Union (aka ‘or’):
set1 | set2

Intersection (aka ‘and’):
set1 & set2

Difference:
set1 - set2

Symmetric Difference (aka ‘xor’):
set1 ^ set2

FOR SEQUENCE TYPES

** # sort by length
list1.sort()

** # sort by length
list1.sort(key = len)

Note: # sort by length is O(n log n) for the sorting algorithm used.

SLICING FOR SEQUENCE TYPES

** # sort by length
list1.sort(key = len)

Note: # sort by length is O(n log n) for the sorting algorithm used.

TUPLE

One dimensional, fixed-length, immutable sequence of Python objects of ANY type.

Note: All non-Get function call i.e. list1.sort() examples below are in-place (without creating a new object) operations unless noted otherwise.
**Functions**

Python is pass by reference, function arguments are passed by reference.

- **Basic Form:**
  ```python
def func1(posArg1, keywordArg1 = None): ...
```

**Note:**
- Keyword arguments MUST follow positional arguments.
- Python by default is NOT "lazy evaluation", expressions are evaluated immediately.

**Function Call Mechanism:**
1. All functions are local to the module level scope. See 'Module' section.
2. Internally, arguments are packed into a tuple and dict, function receives a tuple 'args' and dict 'kwargs' and internally unpack.
3. Common usage of 'Functions are objects':

```python
def func1(posOps = [str, strip, user_define_func, ..], ..):
  for function in ops:
    value = function(value)
```

**RETURN VALUES**
- None is returned if end of function is reached without encountering a return statement.
- Multiple values return via ONE tuple object

```python
return (value1, value2)
value1, value2 = function(value)
```

**Anonymous (aka lambda) Functions**
- What is Anonymous function?
  A simple function consisting of a single statement.

```python
lambda x: x * 2
# def func(x): return x * 2
```

- Application of lambda functions : 'currying' aka deriving new functions from existing ones by partial function application

```python
my0 = lambda x: pd.rolling_mean(x, 60)
```

**Useful Functions** (for data structures)
1. **Enumerate** returns a sequence (i, value) tuples where i is the index of current item.
   ```python
   for i, value in enumerate(collection):
   ```
   - Application : Create a dict mapping of value of a sequence (assumed to be unique) to their locations in the sequence.
2. **Sorted** returns a new sorted list from any sequence
   ```python
   sorted([2, 1, 3]) => [1, 2, 3]
   ```

**Anonymous Functions**
- Application:
  ```python
  sorted(set,['abc', 'bcd']) => ['a', 'b', 'c', 'd']
  # returns sorted unique characters
  ```
3. **Zip** pairs up elements of a number of lists, tuples or other sequences to create a list of tuples:
   ```python
   zip(seq1, seq2) => [(seq1_1, seq2_1), (seq1_2, seq2_2), ..]
   ```
4. **Reversed** iterates over the elements of a sequence in reverse order.
   ```python
   list(reversed(range(10)))
   ```
5. **Useful interactive tool:**
   ```python
   dir(variable1) # list all methods available on the object
   ```

**Control and Flow**
1. **Operators for conditions in 'if else':**
   ```python
   Check if two variables are same object
   var1 == var2
   ```
   - are different object
   ```python
   var1 != var2
   ```
   - Check if two variables have same value
   ```python
   var1 is var2
   ```

**Warning:** Use 'and', 'or', 'not' operators for compound conditions, not & || !

2. **Common usage of 'for operator':**
   ```python
   iterating over a collection (i.e. list or tuple) or an iterator
   for element in iterator:
   ```
   - if elements are sequences, can be 'unpacked'
   ```python
   for a, b, c in iterator:
   ```
   - 'pass' - no-op statement. Used in blocks where no action is to be taken.
   3. **Ternary Expression** - aka less verbose 'if else'
   - **Basic Form:**
   ```python
   value = true-expr if condition
   else false-expr
   ```
   - No switch/case statement, use if/elif instead.

**Object-oriented Programming**
1. 'object' is the root of all Python types
2. Everything (number, string, function, class, module, etc.) is an object, each object has a 'type'. Object variable is a pointer to its location in memory.
3. All objects are reference-counted.
```python
sys.getrefcount() => x
```
- x = 5, b = a
- This creates a 'reference' to the object on the right side of =, thus both a and b point to 5

4. **Class Basic Form:**
   ```python
   class MyObject(object):
   ```
   - '# self' is equivalent of 'this' in Java/C++
   ```python
   def __init__(self, name):
   self.name = name
   ```
   - `@staticmethod`
   ```python
   def classFunc2(name1):
   ```
   - `@classmethod`
   ```python
   def MyObject.classFunc2('b')
   ```

5. **Useful interactive tool:**
   ```python
   dir(MyObject)
   ```

**Common String Operations**
1. **Concatenate List/Tuple with Separator**
   ```python
   [1, 2, 3] + join([10, 20, 30]) => [1, 2, 3, 10, 20, 30]
   ```
2. **Format String**
   ```python
   string1 = 'My name is {0} (name)'
   newString1 = string1.format('Chen')
   ```
3. **Split String**
   ```python
   splitList = string.split(sep)
   ```
4. **Get Substring**
   ```python
   start = 1
   string[start:8] => 'v1, v2, v3'
   ```
5. **String Padding with Zeros**
   ```python
   month = '05'
   month.zfill(2) => '05'
   ```

**Exception Handling**
1. **Basic Form:**
   ```python
   try:
   ... except ValueError as e:
   except (TypeError, AnotherError):
   ```
   - Finally:
   ```python
   ... # cleanup, e.g. close db
   ```

2. **Raise Exception Manually**
   ```python
   raise AssertionError # assertion failed
   raise SystemExit # request program exit
   raise RuntimeError('Error message')
   ```

**List, Set and Dict Comprehensions**
- Syntactic sugar that makes code easier to read and write

1. **List comprehensions**
   - Concisely form a new list by filtering the elements of a collection and transforming the elements passing the filter in one concise expression.
   ```python
   newList = [for val in collection if condition]
   ```
   - A shortcut for:
   ```python
   result = [for val in collection if condition]
   ```
   - The filter condition can be omitted, leaving only the expression.
2. **Dict Comprehension**
   ```python
   {key:expr: value:expr for value in collection if condition}
   ```
   ```python
   {key:val for val in collection if condition}
   ```
3. **Set Comprehension**
   ```python
   {same as List Comprehension except with curly braces instead of []}
   ```
4. **Nestet List Comprehensions**
   ```python
   [for val in collection for innerVal in val if condition]
   ```

**LIST, SET, AND DICT COMPREHENSIONS**

- Formatted string operations
- Concatenate/Join
- Formatting
- Pattern Matching
- Splitting
- Slicing
- Padding

- Exception Handling
- Operator Precedence
- Lambda Functions
- Anonymous Functions
- Useful Functions

- Control Flow
- Ternary Expression

- Classes and Objects
- Inheritance
- Polymorphism
- Method Resolution Order

Created by Ariane Colton and Sean Chen
Based on content from 'Python For Data Analysis' by Wes McKinney
Updated: May 3, 2016