Basic Python Revision Notes

With help from Nitish Mittal

HELP from Documentation

```
dir(module)
help()
```

Important Characters and Sets of Characters

tab	`\t`
new line	`\n`
backslash	`\\`
string	`" "` or `' '`
docstring	`"""   """`
comparison operators	`==` , `<` , `>` , `<=` , `>=` , `!=`
Python type boolean	`True` , `False`.
Logical operators	`not` , `and` , `or`

Order of Operations (from Emory)

Operator	Description
`()`	Parentheses (grouping)
f`(args...)`	Function call
x`[index:index]`	Slicing
x`[index]`	Subscription
x.attribute	Attribute reference
`**`	Exponentiation
+*x*, -*x*	Positive, negative
`*`, `/`, `%`	Multiplication, division, remainder
`+`, `-`	Addition, subtraction
`in`, `not in`, `is`, `is not`, `<`, `<=`, `>`, `>=`, `<>`, `!=`, `==`	Comparisons, membership, identity
`not` *x*	Boolean NOT
`and`	Boolean AND
`or`	Boolean OR

Variable Names

- case sensitive
- cannot start with a number (ex, `1_assd` is not allowed)

Six Steps to Defining a Function

1. What should your function do? Type a couple of example calls.
2. Pick a meaningful name (often a verb or verb phrase): What is a short answer to "What does your function do"?
3. Decide how many parameters the function takes and any return values
4. Describe what your function does and any parameters and return values in the docstring
5. Write the body of the function
6. Test your function. Think about edge cases.

Integers and Strings

```
>>> int(45)
45
>>> int('45')
45
>>> str(45)
'45'
>>> str('45')
'45'
>>> int(str(45))
45
```

Calling Methods

`module_name.function_name(x)`

- `math.sqrt(x)`
- `random.randrange(2,5)`

Conditionals and Branching

- `if`
- `elif`
- `else`

We have a boolean logic expression for `if` which works when the Boolean evaluates to `True`

String Operators

Description	**Operator**	**Example**	**Output**
equality	`==`	`'cat' == 'cat'`	`True`
inequality	`!=`	`'cat' != 'Cat'`	`True`
less than	`<`	`'A' < 'a'`	`True`
greater than	`>`	`'a' > 'A'`	`True`
less than or equal	`<=`	`'a' <= 'a'`	`True`
greater than or equal	`>=`	`'a' >= 'A'`	`True`
contains	`in`	`'cad' in 'abracadabra'`	`True`
length of str s	`len(s)`	`len("abc")`	`3`

String Indexing and Slicing

(`s[a:b]` means index `a` to length `(b-a)` or `a` to `b` index but not including `b`)

- `s[2:3]`
- `s[0]`
- `s[:5]`
- `s[4:]`

String is immutable (ex. `s[4]='a'` will not replace `'a'` and index `4` of `s`)

String Methods

- A method is a function inside of an object.
- The general form of a method call is:
    - object.method(arguments)
    - dir(str)
    - help(str.method)

`for` Loops

```
num_vowels = 0
    for char in s:
        if char in 'aeiouAEIOU':
            num_vowels = num_vowels + 1
    print num_vowels

vowels = ''

for char in s:
    if char in 'aeiouAEIOU':
        vowels = vowels + char
print vowels
```

Lists

Like for strings, slicing and indexing can also be used for lists

```
List = ['a','b',1]
```

• length of list	`len(list)`
• smallest element in list	`min(list)`
• largest element in list	`max(list)`
• sum of elements of list (where list items must be numeric)	`sum(list)`

```
>>> a=[1,'ab',2,'pq']
>>> a[1][0]
'a'
>>> a[1][1]
'b'
>>> a[3][1]
'q'
>>> a[3][2]
```

List Methods

• append a value or string	`list.append('a')`
• extended by another list	`list.extend(['a', 'b'])`

```
>>> a = [5] + [6] + ['a',7]
>>> print (a)
[5, 6, 'a', 7]
```

List Mutability

We say that lists are mutable: they can be modified.

```
>>> lst = [1, 2, 3]
>>> lst[0] = 'apple'
>>> lst
['appple, 2, 3]
```

List Aliasing

Consider the following code:

```
>>> lst1 = [11, 12, 13, 14, 15, 16, 17]
>>> lst2 = lst1
>>> lst1[-1] = 18
>>> lst2
[11, 12, 13, 14, 15, 16, 18]
```

After the second statement executes, `lst1` and `lst2` both refer to the same list. When two variables refer to the same objects, they are aliases. If that list is modified, both of `lst1` and `lst2` will see the change.

But be careful about:

```
>>> lst1 = [11, 12, 13, 14, 15, 16, 17]
>>> lst2 = lst1
>>> lst1 = [5, 6]
>>> lst2
[11, 12, 13, 14, 15, 16, 17]
```

And also:

```
>>> lst1 = [1,2,3]
>>> lst2 = lst1[:]
>>> lst2.remove(2)
>>> lst1
[1,2,3]
```

`while` Loops

```
i = 0
while i < len(s) and not (s[i] in 'aeiouAEIOU'):
    print(s[i])
    i = i + 1

for char in s:
    if not (char in 'aeiouAEIOU'):
        print(char)
```

The difference between the two is that the `for` loop looks at every character in `s`, but the `while` loop ends as soon a vowel is found. So the loops differ on any string where a consonant follows a vowel. `while` is an `if` statement in motion. It is a repeated loop until the boolean test evaluates to `False.`

```
def secret(s):
    i = 0
    result = ''
    while s[i].isdigit():
        result = result + s[i]
        i = i + 1
    print result
```

`>>> secret('123')` will give an error message when it runs the fourth time.

Global and Local Variables

Variables defined outside functions are global variables. Their values may be accessed inside functions without declaration.

To modify to a global variable inside a function, the variable must be declared inside the function using the keyword `global`.

```
def x():
    global num
    num = 5
def y():
    num = 4

>>> num = 7
>>> print (num)
7
>>> x()
>>> print (num)
5
>>> y()
>>> print (num)
5
```

Dictionaries

The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.

- keys can be numbers, strings, Booleans
    - a list is unhashable in a dictionary (cannot be used as a key)
    - a tuple is hashable in a dictionary (can be used as a key).
- values can be dicts, strings, numbers, booleans, lists

```
for key in my_dict:
    value = my_dict[key]
```

This is same as:

```
for key, value in my_dict.items():
```