

Python Notes

Integers:

- any whole number positive or negative

```
int() # takes input and then returns it as an integer
```

Floats:

- floating point numbers

```
float() # takes input and then returns it as a float; 10 becomes 10.0
```

Strings:

- any character in quotations
- are iterable, character by character
- characters are indexed
- triple double quotes allows a string to span multiple lines

```
"""
This is the first line,
and this is the second.
"""
```

- single or double quotes for everything else

```
str() # takes input and then returns it as a string
```

```
len() # returns the number of characters within the string
```

Escape Character:

- "\" is the escape character (backslash)
- using the escape character allows double quotes to be within a string (among other things)

```
print("\nHello there.\n")
# output = "Hello there."
```

- \n adds a new line

Formatted String:

- prefix an 'f' before the quotes of a string to make it a formatted string

- this allows variables to be called and expressions to be entered within the string using {}

```
name = 'John'
print(f"Hello {name}.")
# output = Hello John.
```

.format String:

- older way to use variables and expressions within a string

```
name = 'John'
age = 42
print("{0} is {1} years old.".format(name, age))
# output: John is 42 years old.
```

- the {} can be left empty or indexes can be used as seen in the example above
- using indexes allows a variable to be used multiple times

% Formating:

- another old way to use variables and expression within a string

```
name = 'John'
age = 42
print("%s is %d years old." % (name, age))
# output: John is 42 years old.
```

- %s for strings
- %d for integers

Methods:

```
.upper() # returns a string that is all upper case

.lower() # returns a string that is all lower case

.title() # returns a string in which the first letter in every word is
upper case

.strip() # returns a string with white spaces removed on both sides

.lstrip() .rstrip() # removes white spaces on either the left or right side

.find() # returns the index of specified characters within the string, -1
if the character are not there

.replace()
```

Booleans:

- true or false

```
bool()  # takes input and returns the boolean value
```

- 0 is false while all other numbers are true
 - empty strings and 'None' are also false
-

Lists:

- are defined using []

```
numbers = [1, 2, 3, 4]
```

- values are separated by commas
- items are sorted by index starting at 0

```
enumerate()  # returns a tuple with the index and value of the item
```

Adding/Inserting Items:

```
.append()  # add item to the end of the list
```

```
.insert()  # inserts an item at any index; example:  
.insert(3, 'hello')  # inserts 'hello' at index 3
```

Removing Items:

```
.pop()  # removes item at the end of the list unless an index is specified
```

```
.remove()  # remove the first occurrence of whatever is specified
```

```
.clear()  # removes all of the items
```

Finding Items:

```
.index()  # returns the index of the input; will return an error if the item  
does not exist
```

```
.count()  # returns the number of times something exists within the list
```

Sorting:

```
.sort() sorted()  # sorts in ascending order  
# using reverse=True in () will change it to descending
```

Tuples:

- can be defined with () or a trailing ,

```
tupleA = (2, 3)
tupleB = 4, 5,
```

- items cannot be added or removed; immutable
 - can be concatenated
 - items are indexed
-

Dictionaries:

- are collections of key value pairs
- can be defined by:

```
dictA = {'x': 1, 'y': 2}
dictB = dict(x=1, y=2)
```

- both methods shown in the example above create the same dictionary
- items cannot be looked up by numerical indexes
- keys are used to access values

```
dictA['x']
# returns: 1
```

- keys can be reassigned values;
- if the key does not exist a new key pair will be created

```
dictA['y'] = 7
dictA['z'] = 3
print(dictA)
# output: {'x': 1, 'y': 7, 'z': 3}
```

Methods:

```
.get()
# returns the value of a key
print(dictB.get('x'))
# output: 1
# will return 'None' if the key does not exist; a default value can be
specified
dictB.get('w', 0) # if 'w' does not exist, '0' will be returned
```

```
.values()
# returns a list of the values for every key
```

```
# using list() will clean up the results
```

```
.keys()
```

```
# returns a list of all the keys
```

```
# using list() will clean up the results
```

```
.pop()
```

```
# removes a key vlaue pair
```

```
# will take the value of the key being removed
```

```
dictB.pop('y') # 'y' will be removed and its value will be assigned to  
'value'
```

Math:

Division:

- / returns a float
- // returns an integer
- % returns the modulus (the remainder in division)

```
20 / 3 = 6.7
```

```
20 // 3 = 6
```

```
20 % = 2
```

Exponent:

```
2 ** 3 = 8
```

```
# 2 to the power of 3
```

Basic Functions:

```
round() # rounds the number to the closest integer unless a second number  
is entered to specify decimal place
```

```
abs() # returns the absolute value
```

Complex Numbers:

- are imaginary numbers in math
- are represented as 'j' in Python while in math they are represented as 'i'

Operators:

Logical:

- and
- or

- not

Comparison:

```
<
<=
>
>=
== equal to
!= not equal to
```

Chained Comparison:

```
18 <= age < 65
# this would replace:
age >= 18 and age < 65
```