Python Basics

Variables and types

Variables are assigned a type by default!

```
my_variable_name = 3
my_variable_name2 = 3.2
my_variable_name3 = 'A string!'
```

Checking types:

```
type(my_variable)
# But.... checking for a type:
if isinstance(my_variable, int):
    print('It is an Integer!')
    # or if not....
# isinstance supports inheritance
```

Converting to another type

```
to_string = str(3.2)
my_int = 7
to_string2 = str(my_int)
```

What if you can't convert ('a' to int('a') => ValueError!)

Keywords:

```
# False, None, True
a = True
if a is not False:
    b = None  # b is assigned None, a special constant representing the 

# and, as
if a and b is None:  # checking if a is True and b is None
    c = a as bool  # 'as' is usually used in 'with' statements or import

# assert
assert a == True, "a should be True"  # raises an AssertionError if the 
```

```
# break
for i in range(10):
    if i == 5:
        break  # exit the loop

# class
class MyClass:
    pass

# continue
for i in range(10):
    if i % 2 == 0:
        continue  # skip the rest of the loop for even numbers
    print(i)

# def
def my_function():
    return True

# del
my_list = [1, 2, 3]
del my_list[1]  # removes the item at index 1

# elif, else
if a is False:
    print("a is False")
elif a is True:
    print("a is True")
else:
    print("a is neither True nor False")

# except, finally, try
try:
    1 / 0
except ZeroDivisionError:
    print("Can't divide by zero!")
finally:
    print("This will execute no matter what")

# for, from, in
for i in range(3):
    pass  # will pass through this loop 3 times

# if
if a == True:
    pass
```

```python
# import, global
import math
global x
x = math.pi

# is
if type(a) is bool:
    pass

# lambda
add = lambda x, y: x + y

# nonlocal
def outer():
    x = "local"
    def inner():
        nonlocal x
        x = "nonlocal"
    inner()
    return x

# not
if not a:
    pass

# or
if a == True or b is None:
    pass

# pass
def func():
    pass  # a function that does nothing (yet)

# raise
raise ValueError("This is an error message")

# return
def return_function():
    return True

# while
while a is True:
    a = False  # this loop will run once

# with
with open('file.txt', 'w') as f:
    f.write('Hello, world!')
```

```
# yield
def generator():
    yield 'Hello'
    yield 'World'
```

Operators

Only special: "//" and "%"

Mutability!!!

int, str, tuple: immutable list, set: mutable

copy.deepcopy()!

Nooby habits

```
my_number = 3.721839823
print(f"My number is {round(my_number, 2)}")
print(f"My number is {my_number:.f2}")
print("My number is (.f2)".format(my_number))
```

```
# using too many global variables in functions
# single letter variables
# and/or in print statement
# use divmod()
```

```
# insert or deleting during iteration
# use a r"" path!

if __name__ == '__main__':
    pass
    # do something!
```

```
# using string+string instead of f"..."
```

```
 f = open(my-file_path, "w")
 f.write("hi\n")
 f.close

 with open(my_file_path, "w") as f:
    f.write("hi\n")
```

```
try:
    # something
except Exception as e:
    print(e)
```

```
# don"t initiate a list in a function
def my_function(n, my_list = []):
    ....
# every call will share the same list!
def my_function(n, my_list = None):
    if my_list is None:
        my_list = [] # or list()
```

```
if bool(x):
    pass
if len(x) != 0:
    pass
if x:
    pass
```

```
for i in range(len(my_list)):
    value = my_list[i]

for a, b in zip(my_a, my_b):
    pass

for key in d.keys():
    pass

for key in list(d.keys()):
    pass

for key in d:
    pass

for key, value in d.items():
    pass
```