Lab 12
Dexter Kale – CS 3035 – 05

1. Different Data Types

```
>>> number = 1
>>> string = 'hi'
>>> list = [1. 'hi', 3.4]
  File "<stdin>", line 1
    list = [1. 'hi', 3.4]
              ^^^^^^^
SyntaxError: invalid syntax. Perhaps you forgot a comma?
>>> list = [1, 'hi', 3.4]
>>> dict = ['hey': 'bye', 'hungry': 'full']
  File "<stdin>", line 1
    dict = ['hey': 'bye', 'hungry': 'full']
                 ^
SyntaxError: invalid syntax
>>> dict = {'hey': 'bye', 'hungry': 'full'}
>>> type(int)
<class 'type'>
>>> type(string)
<class 'str'>
>>> type(list)
<class 'list'>
>>> type(dict)
<class 'dict'>
>>>
```

Note:type(int) wasn't done correctly

```
>>> num = 1
>>> type(num)
<class 'int'>
>>>
```

2.

```
>>> string = "hi"
>>> num = 5
>>> num + num
10
>>> num - num
0
>>> num * num
25
>>> num / num
1.0
>>> num ** num
3125
>>>
```

```
>>> string + string
'hihi'
>>> string - string
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for -: 'str' and 'str'
>>> string * string
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can't multiply sequence by non-int of type 'str'
>>> string /string
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for /: 'str' and 'str'
>>> string ** string
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'str'
>>>
```

In conclusion:
int, string,
can both add but
String can't do any
other operation, while
int can do them all.

b.

```
>>> num + string
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> num - string
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for -: 'int' and 'str'
>>> num * string
'hihihihihi'
>>> num / string
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for /: 'int' and 'str'
>>> num ** string
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for ** or pow(): 'int' and 'str'
>>> string ** num
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'int'
>>>
```

Only num * string seems to work, duplicating it. Every other function doesn't work, resulting in a type error.

Num * string's output will be a string.

3.
List: .len(), .insert(), .append()
len() returns how many elements are in the list.
Insert will likely put in an element at the specified location.
append will likely be able to add on a new element at the end.

```
>>> l.append(4)
>>> print(l)
[1, 2, 3, 4]
>>> l.insert(0, 10)
>>> print(l)
[10, 1, 2, 3, 4]
>>>
>>> len(l)
3
```

Append does in fact put elements in the end

Insert does put elements in the requested index.

Length does return the length of it's List.

String:

isdigit() probably just returns a boolean if it's a number of any kind.
count() probably just returns the number of times something appears in that string
join() probably just combines 2 strings together.

```
>>> str = 'hi'
>>> str.isdigit()
False
>>> str.count('hi')
1
>>> str.join(' my name is Dexter')
' himhiyhi hinhiahimhiehi hiihishi hiDhiehixhithiehir'
>>>
```

isDigit did return a boolean if the string was a digit.
Count did count the instances of a string in the string.

Join however did something that I didn't expect at all. It seems to have instead put in the string at every instance of the new string, character by character.

Dictionary:

get() probably gets the requested item if the correct keyword was used.
Keys() probably prints all the keys.
Pop() probably removes and returns the element, with the correct key used.

```
>>> d.get('hi')
'bye'
>>> d.keys()
dict_keys(['apple', 'hi'])
>>> d.pop('hi')
'bye'
>>> d.keys()
dict_keys(['apple'])
>>>
```

.get does get the requested element based on given key
.keys does return all keywords available in the dictionary
.pop does remove the element with the key given.

4.
floor() likely returns the number that's up to the lower integer.
Ceil() likely returns the number that's up to the higher integer.
Factorial() likely returns the factorial of the number given.

```
>>> x = 654.3048
>>> math.floor(x)
654
>>> math.ceil(x)
655
>>> x = 655
>>> math.factorial(x)
960354743658084575444214898446561792984428632587403551801806138599793129
914892066509357423586090147568852437443338433416855710924111171481744777
762778611516776593730276757074986996944433515875171122685697615434709280
253368774159180705405267453794051060838374123058469606225244108791475810
977053375012388214784827276632485214599677717615509544550447773645948700
243311843471168916752481379277597943737796515466207014546882331689796610
079527773512381250588269236169550125953266264370444264052513838985501020
607867563166732590965021242591138228887240116382720797567280373414415350
588681424218127534460032760204662115135153069447442688837533353622530720
851014459288250860711600634473551961530321177156125955581175283481864840
293124425084290098104018497321123595565098832747792555882039037106407660
869165631591672258915073662512320264403219531662760391144534759406073560
279658490536344514428671544275200386235591930293951944797286350429069920
828858868885065956245002129393425502652746272122251270192263014908436420
959580812929413530517104495528408637330450929924380824975931737047431260
742343825536426673833894741638856730267498748848005801113157778905424190
329731462929307264285755217901371742819148199125292777534678601874608080
057717618984070305377696407981622164331154656498701738553597318096319110
100184461848477805953968105026138455245815940334165451991844258997082280
395198018681560604997615072697804828992740188487680000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
```

Floor does return the given float's lower bounded integer
Ceil does return the given float's higher bounded integer
Factorial does return the given integer's factorial.

Dynamic/Strong typing generally doesn't have too many differences when it comes to how the module works.

5.
It's not completely true, as string can be added and multiplied together, but is just a concatenation.

```
>>> str = str + str
>>> print(str)
hihi
```

As we see here, it is mutable in being able to change the values directly.