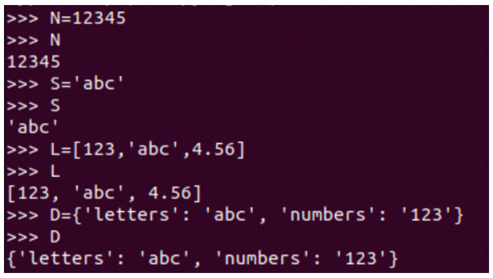
1. Experiment with different data types using the Python interpreter shell. Create a data object of each type: number, string, list, dictionary. Check the data type shown for each type. Take a screenshot of your answer.



```
>>> N=12345
>>> N
12345
>>> S='abc'
>>> S
'abc'
>>> L=[123,'abc',4.56]
>>> L
[123, 'abc', 4.56]
>>> D={'letters': 'abc', 'numbers': '123'}
>>> D
{'letters': 'abc', 'numbers': '123'}
```

2. Python uses expression operators +, -, *, /, **. These mean different things depending on data types.

A. Check which of these expression operators apply to numbers vs. strings. What is the difference in what they do for each data type? Write your observations.

numbers-to-numbers:
All the operators apply when both operands are numbers.
When both operands were integers, the outputs to each of the operators returned an integer value a well, except for the division operator. When I did 4/2, my answer was a float 2.0.
When both operands were floats then all the outputs returned as a floating number.

strings-to-strings:
The only operator that worked was the addition operator +. All other operators with string operands produced a type error output that stated that it's an unsupported operand type.

B. Check the typing rules that apply to these expressions when you apply them to expressions containing numbers and strings. Write your observations.

strings-to-numbers:
The only operator that works for both strings and numbers is the multiplication operator *.
For example, 'abc'*3 produces an output of 'abcabcabc'. Here, the * operator concatenates the strings three times.
However, the number value must be an integer to avoid any errors. Typing 'abc'*0 produces '', an empty string.

3. Python includes methods that are applicable to objects of different data types. For a list L or a string S in Python, you can list the object methods available using dir(L) or dir(S). More details of each method can be found using the help() function. The help function accepts a data type as an argument and returns all associated in-built methods. To get out of the help() function, press 'q'. Make sure you use the data type name as listed in the output of type() as an argument to the help function

A. Note down at least 3 methods of your choice for strings, dictionaries, and lists. You can ignore methods with leading and trailing double underscores. These are used to represent implementations of objects and support customization. The names without underscores are callable methods.
Three methods for dictionaries: get(), keys(), pop().

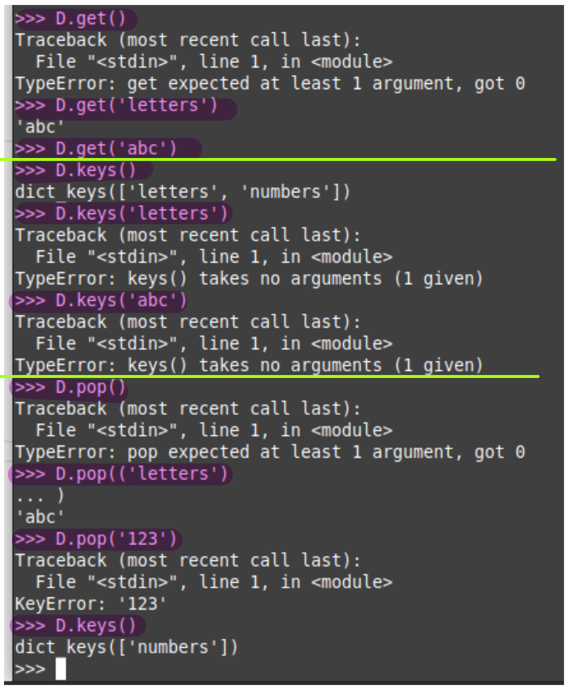
B. What do you think these methods should do? Write in 1-2 lines for each method.
get(): returns the value of a dictionary when the corresponding key is entered.
keys(): returns the keys of a given dictionary.
pop(): pops either the first or last key-value pair from a dictionary.

C. Do some quick testing in the Python interpreter shell to see if your methods of choice did what you expected them to do. Write your observations and show a screenshot of applied methods.

get() only returns the value when the key is entered. It will not work without a key as the input. Other a type error will occur.

keys() returns the list of keys from the dictionary without the values. There is no input required within the parenthesis.
Type error occur as well and states it requires no arguments.

pop() does not pop either the first or last key-value pair from the dictionary. Instead, it requires a key in order to remove the key-value pair from the dictionary.



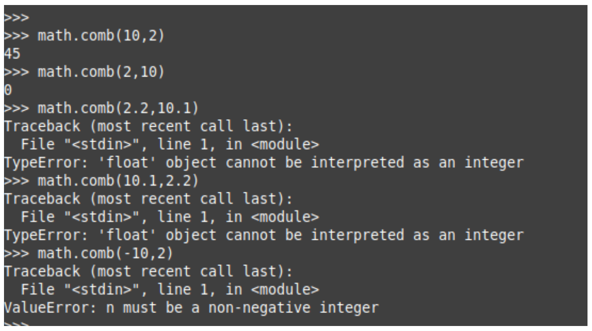
```
>>> D.get()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: get expected at least 1 argument, got 0
>>> D.get('letters')
'abc'
>>> D.get('abc')
>>> D.keys()
dict_keys(['letters', 'numbers'])
>>> D.keys('letters')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: keys() takes no arguments (1 given)
>>> D.keys('abc')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: keys() takes no arguments (1 given)
>>> D.pop()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: pop expected at least 1 argument, got 0
>>> D.pop(('letters')
... )
'abc'
>>> D.pop('123')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: '123'
>>> D.keys()
dict_keys(['numbers'])
>>> 
```

4. Python also includes modules like the Math module and the Random module. To load the math module, do import(math). Use dir(math) to check the methods within the Math module. Math capabilities can be run as math.<method_name>. For example, the Math module includes a 'pow' method. To use this method, you can run math.pow(x,y)where x and y are the numbers of your choice. To find out more about a method, use the help function. For example, help(math.pow) tells you what the pow function in the math module does.

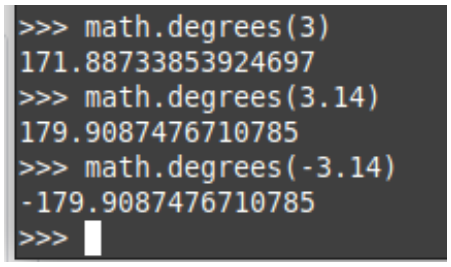
A. Note down 5 methods within the math module and write down what you think these methods should do.
comb(): not sure. Probably combines and returns some values.
degrees(): returns the degrees of an input.
exp(): returns the value of e^x where exp takes the input for x.
fabs(): returns the absolute value of a number.

B. Do some quick testing in the Python interpreter shell to see if your methods of choice did what you expected them to do. Write your observations and show a screenshot of applied methods.

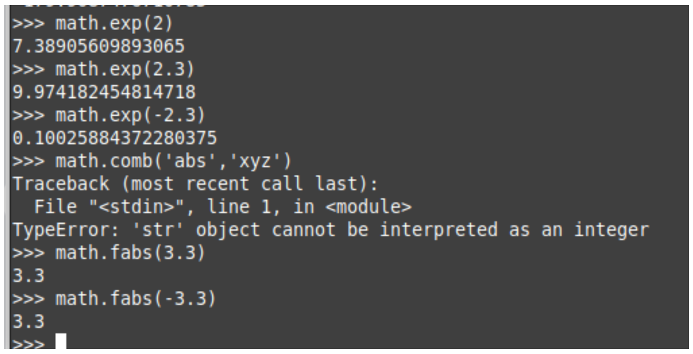
comb(n,k) returns the number of ways to choose k items from a sample size of n items without repetition and without order.
degrees(x): converts angle x from radians to degrees.
My guess of exp() and fabs() seemed to have the correct actions.
For comb(), the first argument must be greater than the second argument. The arguments must also be non-negative integers.
degrees(), exp() and fabs() can accept integers, floats and negative values.



```
>>>
>>> math.comb(10,2)
45
>>> math.comb(2,10)
0
>>> math.comb(2.2,10.1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'float' object cannot be interpreted as an integer
>>> math.comb(10.1,2.2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'float' object cannot be interpreted as an integer
>>> math.comb(-10,2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: n must be a non-negative integer
```



```
>>> math.degrees(3)
171.88733853924697
>>> math.degrees(3.14)
179.9087476710785
>>> math.degrees(-3.14)
-179.9087476710785
>>> 
```



```
>>> math.exp(2)
7.38905609893065
>>> math.exp(2.3)
9.974182454814718
>>> math.exp(-2.3)
0.10025884372280375
>>> math.comb('abs','xyz')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object cannot be interpreted as an integer
>>> math.fabs(3.3)
3.3
>>> math.fabs(-3.3)
3.3
>>> 
```

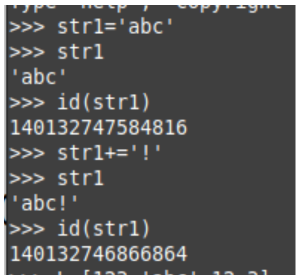
C. Observe what dynamic typing / strongly typing rules apply to different types. For example, try using the math module methods on different data types and check the type of the result. Write down a short summary of your observations. We will discuss these observations in class.

For fabs(), exp() and degrees(), arguments of string and char give a type error stating that the arguments must be a real number. For comb(), the same type of arguments returns a name error saying that the input is not defined.

5. Strings are immutable while lists and dictionaries are mutable. Do some quick testing in the Python interpreter shell to test this claim. Does this hold true for the cases you tried? Show a screenshot of your results.

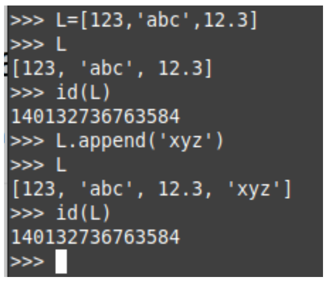
Yes, strings are immutable and lists and dictionaries are mutable.
The memory address of string variables change when another string is added to the original string.
However, when lists and dictionaries are appended or updated, the memory address remains the same.

Strings



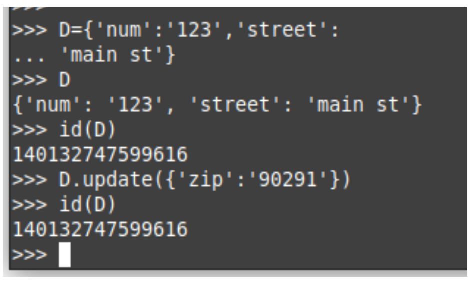
```
>>> str1='abc'
>>> str1
'abc'
>>> id(str1)
140132747584816
>>> str1+='!'
>>> str1
'abc!'
>>> id(str1)
140132746866864
```

List



```
>>> L=[123,'abc',12.3]
>>> L
[123, 'abc', 12.3]
>>> id(L)
140132736763584
>>> L.append('xyz')
>>> L
[123, 'abc', 12.3, 'xyz']
>>> id(L)
140132736763584
>>> 
```

Dictionaries



```
>>> D={'num':'123','street':
... 'main st'}
>>> D
{'num': '123', 'street': 'main st'}
>>> id(D)
140132747599616
>>> D.update({'zip':'90291'})
>>> id(D)
140132747599616
>>> 
```