Python Primer 101

AL NAFI,
A company with a focus on education, wellbeing and renewable energy.

الْحَمْـدُ للهِ الَّذي بِنِـعْمَتِهِ تَتِـمُّ الصّـالِحات

All Praise is for Allah by whose favor good works are accomplished.

الْحَمْـدُ للهِ على كُـلِّ حال

All Praise is for Allah in all circumstances."

Study, Rinse and Repeat

- Please subscribe to our YouTube Channel to be on top of your studies.
- Please keep an eye on zone@alnafi.com emails
- Please review the videos of 30 minutes daily
- Please review the notes daily.
- Please take time for clearing up your mind and reflect on how things are proceeding in your studies.

Good morning, Class.
Today we will be coding in Python.

Strings in Python

- When programming, we usually call text as ***String.***
- Think of string as a collection of letters, and the terms makes sense.
- All the letters and text in this presentation is a ***string***

Creating strings

We create a string by putting quotes around text because programming languages.

We need to tell a computer whether a value is a number, a string or something else.

```
abubakr = 'first caliph!!'
print(abubakr)
```

Creating strings continued…

One can also use double quotes

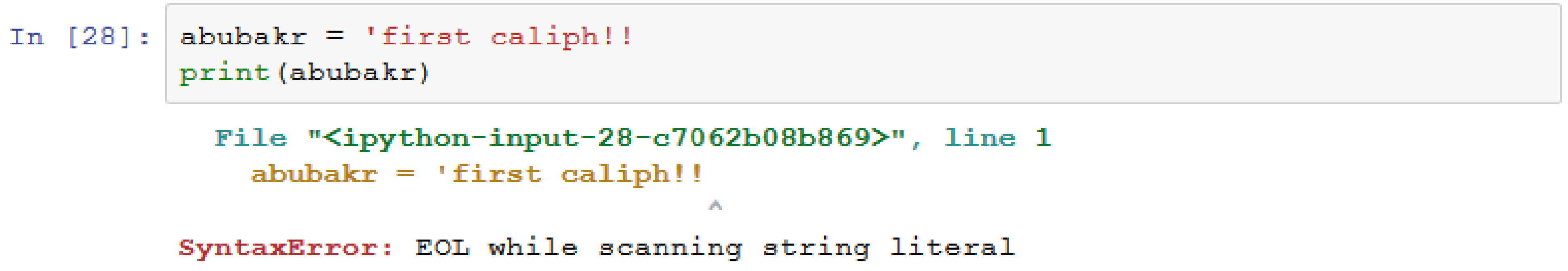
```
abubakr = “first caliph!!”
print(abubakr)
```

But you cannot start with single quote and end with double quote as you will get an error message.

It has to be consistent.

Syntax and Syntax Error

- Syntax means the arrangement and order of word in a sentence
- Syntax error means that you did something in an order python was not expecting or you missed something.
- EOL means end of line



```
In [28]: abubakr = 'first caliph!!
         print(abubakr)

           File "<ipython-input-28-c7062b08b869>", line 1
             abubakr = 'first caliph!!
                                       ^

         SyntaxError: EOL while scanning string literal
```

Adding second line to your command

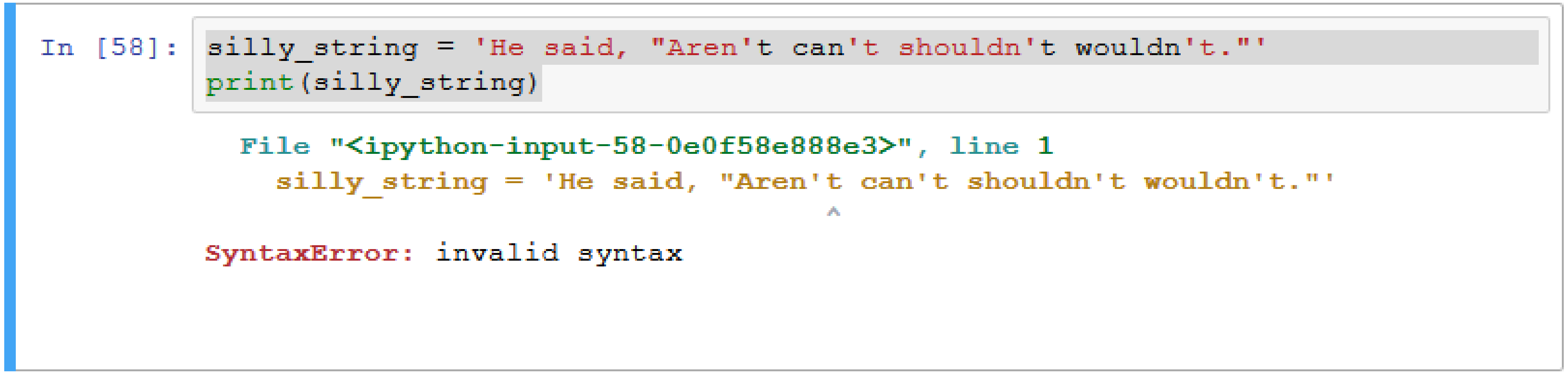
- You can use 3 single quotes to write a second or third or as many lines as you want to a given command.

```
abubakr = ''first
Righteous
caliph!!''
print(abubakr)
```

Strings Problems handling

silly_string = 'He said, "Aren't can't shouldn't wouldn't."'

print(silly_string)

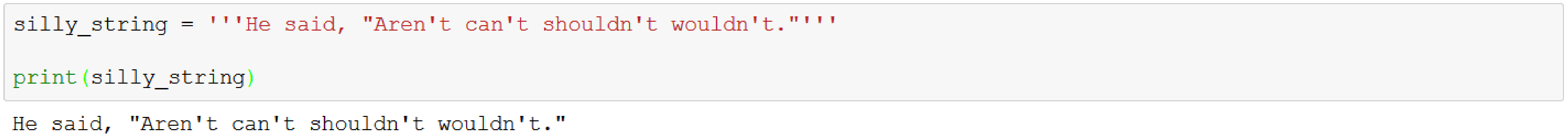


```
In [58]: silly_string = 'He said, "Aren't can't shouldn't wouldn't."'
         print(silly_string)

           File "<ipython-input-58-0e0f58e888e3>", line 1
             silly_string = 'He said, "Aren't can't shouldn't wouldn't."'
                                           ^
         SyntaxError: invalid syntax
```

When Python sees a quotation mark (either a single or double quote), it expects a string to start following the first mark and the string to end after the next matching quotation mark (either single or double) on that line.

```
silly_string = '''He said, "Aren't can't shouldn't wouldn't."'''
print(silly_string)
```



```
silly_string = '''He said, "Aren't can't shouldn't wouldn't."'''

print(silly_string)
```

```
He said, "Aren't can't shouldn't wouldn't."
```

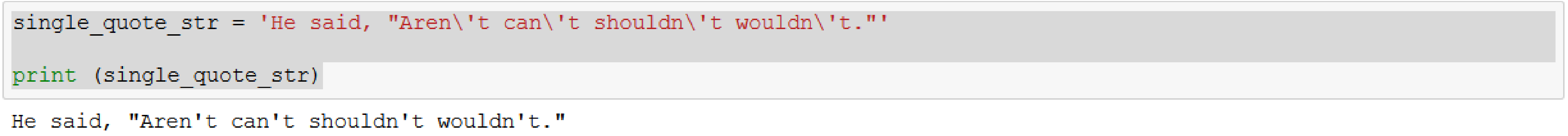
Adding a backslash \ or escaping

\ or excaping means “Yes, I know I have quotes inside my string, and I want you to ignore them until you see the end quote.”

Escaping strings can make them harder to read, so it’s probably better to use multiline strings. Still, you might come across snippets of code that use escaping, so it’s good to know why the backslashes are there. Here are a few examples of how escaping works:

single_quote_str = 'He said, "Aren\'t can\'t shouldn\'t wouldn\'t."'

print (single_quote_str)



```
single_quote_str = 'He said, "Aren\'t can\'t shouldn\'t wouldn\'t."'

print (single_quote_str)
```

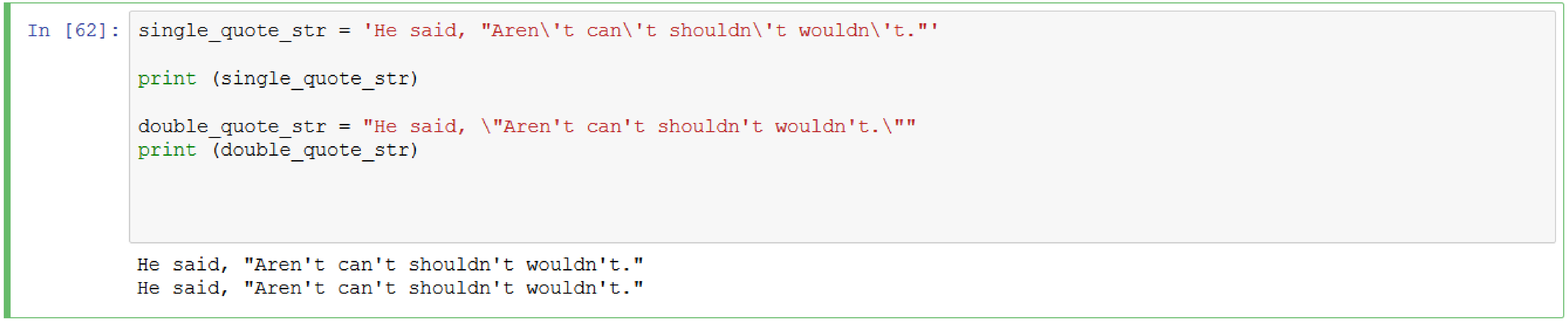
```
He said, "Aren't can't shouldn't wouldn't."
```

single_quote_str = 'He said, "Aren\'t can\'t shouldn\'t wouldn\'t."'

print (single_quote_str)

double_quote_str = "He said, \"Aren't can't shouldn't wouldn't.\""

print (double_quote_str)



```
In [62]: single_quote_str = 'He said, "Aren\'t can\'t shouldn\'t wouldn\'t."'

         print (single_quote_str)

         double_quote_str = "He said, \"Aren't can't shouldn't wouldn't.\""
         print (double_quote_str)
```

```
He said, "Aren't can't shouldn't wouldn't."
He said, "Aren't can't shouldn't wouldn't."
```

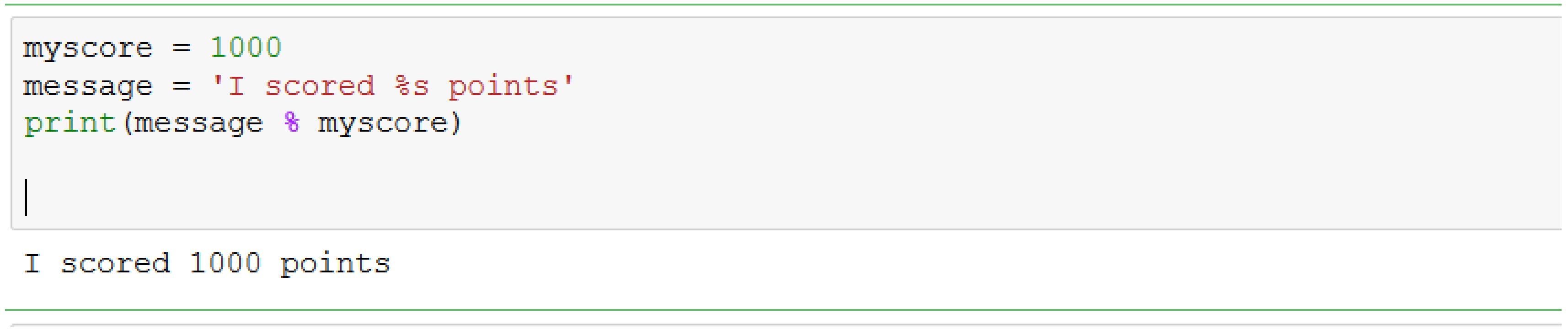
Embedding value in strings

(Embedding values, also referred to as string substitution, is programmer-speak for "inserting values.") For example, to have Python calculate or store the number of points you scored in a game, and then add it to a sentence like "I scored points," use %s in the sentence in place of the value, and then tell Python that value, like this:

myscore = 1000

message = 'I scored %s points'

print(message % myscore)



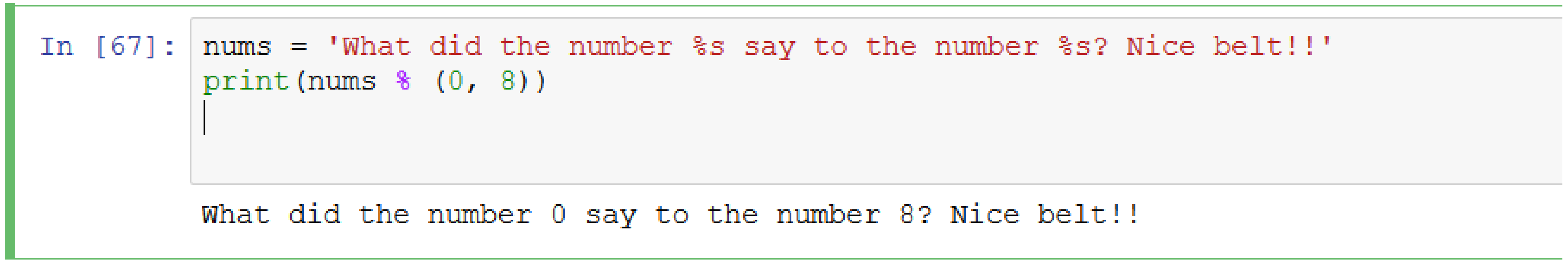
```
myscore = 1000
message = 'I scored %s points'
print(message % myscore)
```

```
I scored 1000 points
```

Embedding value in strings continued...

nums = 'What did the number %s say to the number %s? Nice belt!!'

print(nums % (0, 8))



```
In [67]: nums = 'What did the number %s say to the number %s? Nice belt!!'
         print(nums % (0, 8))

         What did the number 0 say to the number 8? Nice belt!!
```

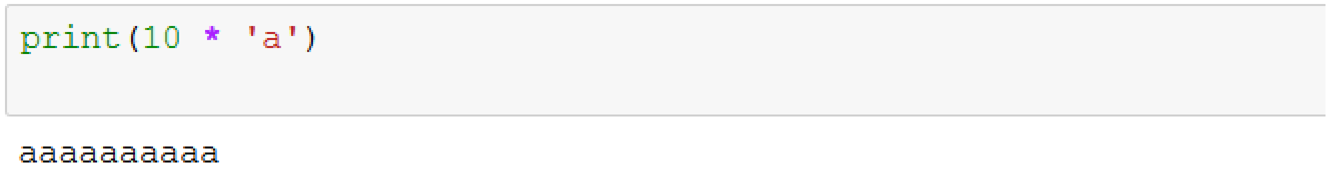
When using more than one placeholder, be sure to wrap the replacement values in parentheses, as shown in the example. The order of the values is the order in which they'll be used in the string.

Multiplying Strings

What is 10 multiplied by 5? The answer is 50, of course. But what’s 10 multiplied by a? Here’s Python’s answer:

print(10 * 'a')

aaaaaaaaaa



```
print(10 * 'a')
```

```
aaaaaaaaaa
```

Multiplying Strings continued

```
spaces = ' ' * 25
print('%s 12 DHA Phase 5' % spaces)
print('%s Clifton' % spaces)
print('%s West Snoring' % spaces)
print()
print()
print('Dear Sir')
print()
print('I wish to report that tiles are missing from the')
print('outside toilet roof.')
print('I think it was bad wind the other night that blew them away.')
print()
print('Regards')
print('Sharfoo')
```

Fun Stuff

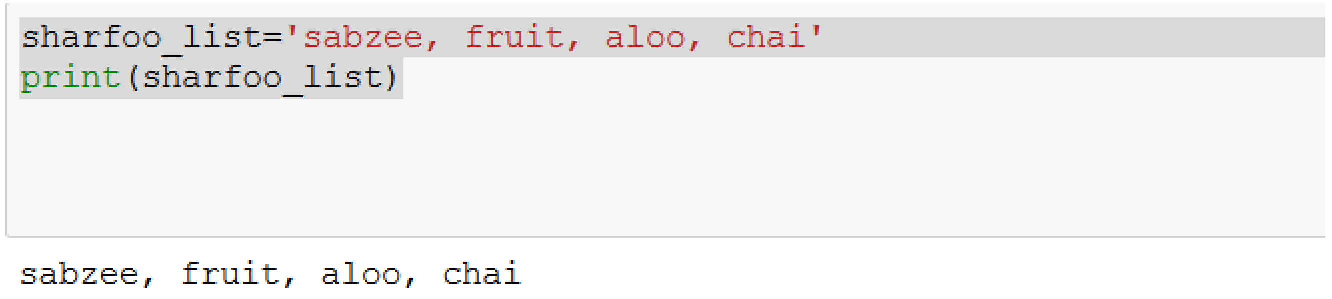
Try this code and share what happens

```
print(1000 * 'Karachi')
```

Lists vs. string

sharfoo_list='sabzee, fruit, aloo, chai'

print(sharfoo_list)



```
sharfoo_list='sabzee, fruit, aloo, chai'
print(sharfoo_list)
```

```
sabzee, fruit, aloo, chai
```

Creating a list

```
sharfoo_list=['sabzee, fruit, aloo, chai']
print(sharfoo_list[2])
```

Creating a list takes a bit more typing than creating a string, but a list is more useful than a string because it can be manipulated.

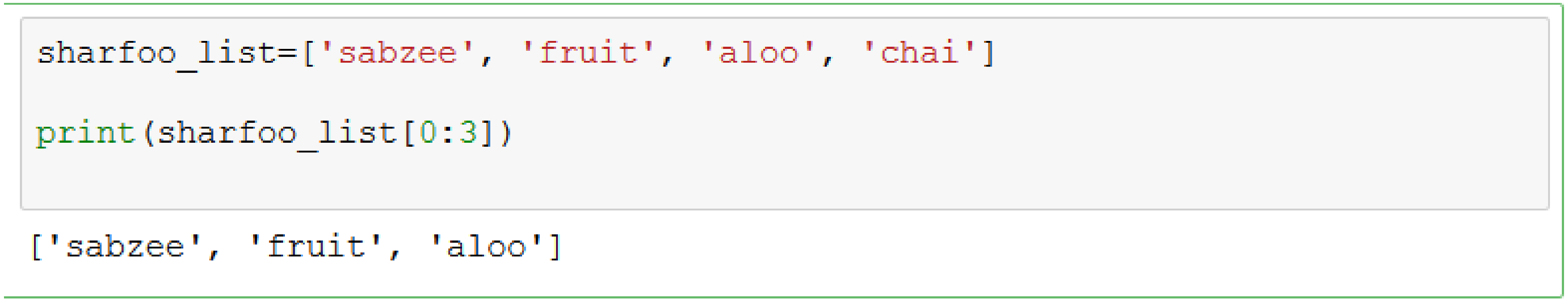
For example we can print the third item in the list.

```
sharfoo_list=['sabzee', 'fruit', 'aloo', 'chai']

print(sharfoo_list[2])
```

sharfoo_list=['sabzee', 'fruit', 'aloo', 'chai']

print(sharfoo_list[0:3])



```
sharfoo_list=['sabzee', 'fruit', 'aloo', 'chai']

print(sharfoo_list[0:3])
```

```
['sabzee', 'fruit', 'aloo']
```

Writing [0:3] is the same as saying, “show the items from index position 0 up to (but not including) index position 3”—or in other words, items 0, 1, and 2.

Lists can store all sorts of items, like numbers

```
some_numbers = [1, 2, 5, 10, 20]
```

They can also hold strings:

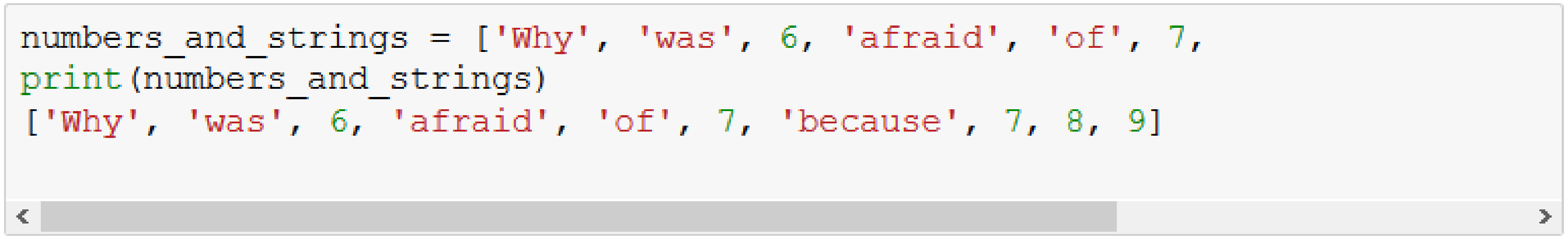
```
some_strings = ['Which', 'Witch', 'Is', 'Which']
```

List strings and number example

numbers_and_strings = ['Why', 'was', 6, 'afraid', 'of', 7, 'because', 7, 8, 9]

print(numbers_and_strings)

['Why', 'was', 6, 'afraid', 'of', 7, 'because', 7, 8, 9]



```
numbers_and_strings = ['Why', 'was', 6, 'afraid', 'of', 7,
print(numbers_and_strings)
['Why', 'was', 6, 'afraid', 'of', 7, 'because', 7, 8, 9]
```

```
['Why', 'was', 6, 'afraid', 'of', 7, 'because', 7, 8, 9]

['Why', 'was', 6, 'afraid', 'of', 7, 'because', 7, 8, 9]
```

List within a list ☺

```
numbers = [1, 2, 3, 4]
strings = ['I', 'kicked', 'my', 'toe', 'and', 'it', 'is', 'sore']
mylist = [numbers, strings]
print(mylist)
[[1, 2, 3, 4], ['I', 'kicked', 'my', 'toe', 'and', 'it', 'is', 'sore']]
```

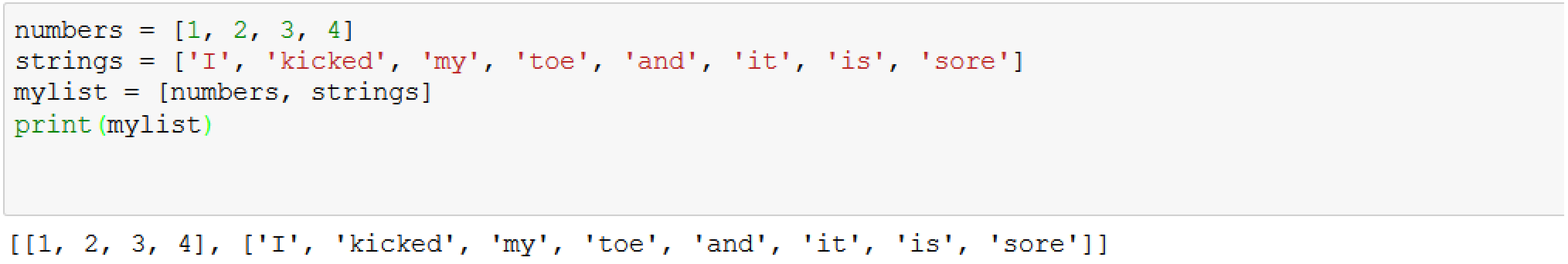
Adding items to a LIST

numbers = [1, 2, 3, 4]

strings = ['I', 'kicked', 'my', 'toe', 'and', 'it', 'is', 'sore']

mylist = [numbers, strings]

print(mylist)



```
numbers = [1, 2, 3, 4]
strings = ['I', 'kicked', 'my', 'toe', 'and', 'it', 'is', 'sore']
mylist = [numbers, strings]
print(mylist)
```

```
[[1, 2, 3, 4], ['I', 'kicked', 'my', 'toe', 'and', 'it', 'is', 'sore']]
```

Adding items to a list

Before adding an item to a list:

```
wizard_list = ['spider legs', 'toe of frog', 'eye of newt', 'bat wing', 'slug butter', 'snake dandruff']
print(wizard_list)
```

After adding item to a list via .append

```
wizard_list = ['spider legs', 'toe of frog', 'eye of newt', 'bat wing', 'slug butter', 'snake dandruff']
wizard_list.append('bear burp')
print(wizard_list)
```

Challenge!

Now add the following to the previous list and print.

```
wizard_list.append('mandrake')
wizard_list.append('hemlock')
wizard_list.append('swamp gas')
```

Removing items from the list

To add an item to a list we use **append**

To remove an item from a list we use **del** short for delete

```
wizard_list = ['spider legs', 'toe of frog', 'eye of newt', 'bat wing', 'slug
butter', 'snake dandruff']
wizard_list.append('bear burp')
del wizard_list [5]
print(wizard_list)
```

Remember that positions start at zero, so `wiz-ard_list[5]` *actually refers to the sixth item in the list.*

List Arithmetic

We can join two lists by using +

```
list1 = [1, 2, 3, 4]
list2 = ['I', 'tripped', 'over', 'and', 'hit', 'the', 'floor']
print(list1 + list2)
```

We can also add the two lists and set the result equal to another variable.

```
list1 = [1, 2, 3, 4]
list2 = ['I', 'ate', 'chocolate', 'and', 'I', 'want', 'more']
list3 = list1 + list2
print(list3)
```

And we can multiply a list by a number. For example, to multiply list1 by 5, we write list1 * 5:

```
list1 = [1, 2]
print(list1 * 5)
```

```
[1, 2, 1, 2, 1, 2, 1, 2, 1, 2]
```

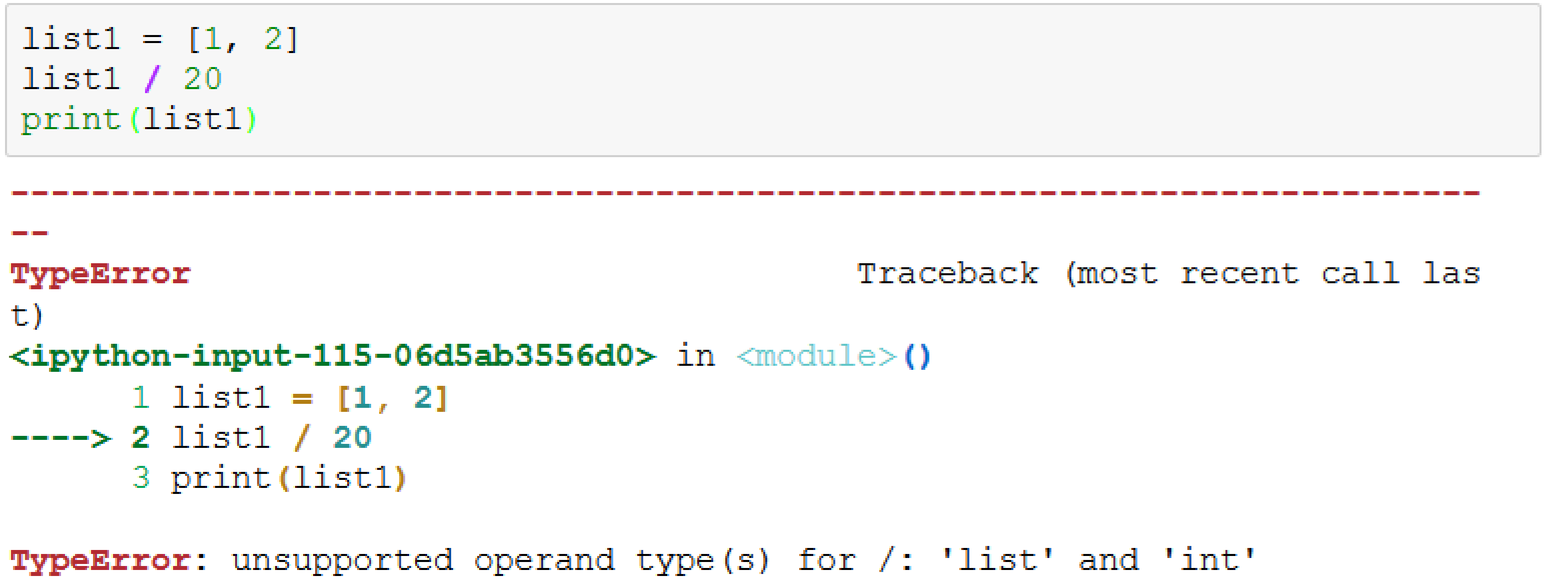
This is actually telling Python to repeat list1 five times, resulting in 1, 2, 1, 2, 1, 2, 1, 2, 1, 2.

On the other hand, division (/) and subtraction (-) give only errors, as in these examples:

list1 = [1, 2]

list1 / 20

print(list1)

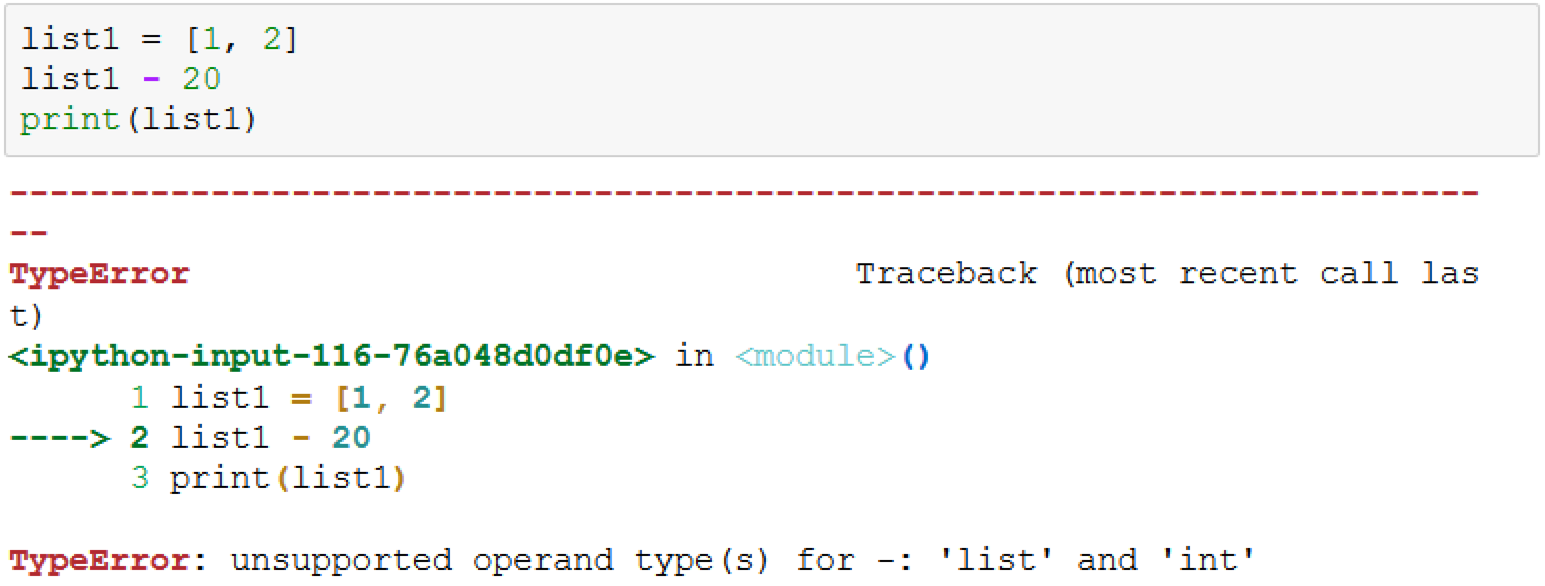


```
list1 = [1, 2]
list1 / 20
print(list1)
```

```
---------------------------------------------------------------------------
--
TypeError                                 Traceback (most recent call las
t)
<ipython-input-115-06d5ab3556d0> in <module>()
      1 list1 = [1, 2]
----> 2 list1 / 20
      3 print(list1)

TypeError: unsupported operand type(s) for /: 'list' and 'int'
```

```
list1 = [1, 2]
list1 - 20
print(list1)
```



```
list1 = [1, 2]
list1 - 20
print(list1)
```

```
---------------------------------------------------------------------------
--
TypeError                                 Traceback (most recent call las
t)
<ipython-input-116-76a048d0df0e> in <module>()
      1 list1 = [1, 2]
----> 2 list1 - 20
      3 print(list1)

TypeError: unsupported operand type(s) for -: 'list' and 'int'
```

But why? Well, joining lists with + and repeating lists with * are straightforward enough operations.

These concepts also make sense in the real world. For example, if I were to hand you two paper shopping lists and say, “Add these two lists,” you might write out all the items on another sheet of paper in order, end to end.

The same might be true if I said, “Multiply this list by 3.” You could imagine writing a list of all of the list’s items three times on another sheet of paper. But how would you divide a list? For example, consider how you would divide a list of six numbers (1 through 6) in two. Here are just three different ways:

[1, 2, 3]	[4, 5, 6]
[1]	[2, 3, 4, 5, 6]
[1, 2, 3, 4]	[5, 6]

Would we divide the list in the middle, split it after the first item, or just pick some random place and divide it there? There’s no simple answer, and when you ask Python to divide a list, it doesn’t know what to do, either. That’s why it responds with an error.

Tuples

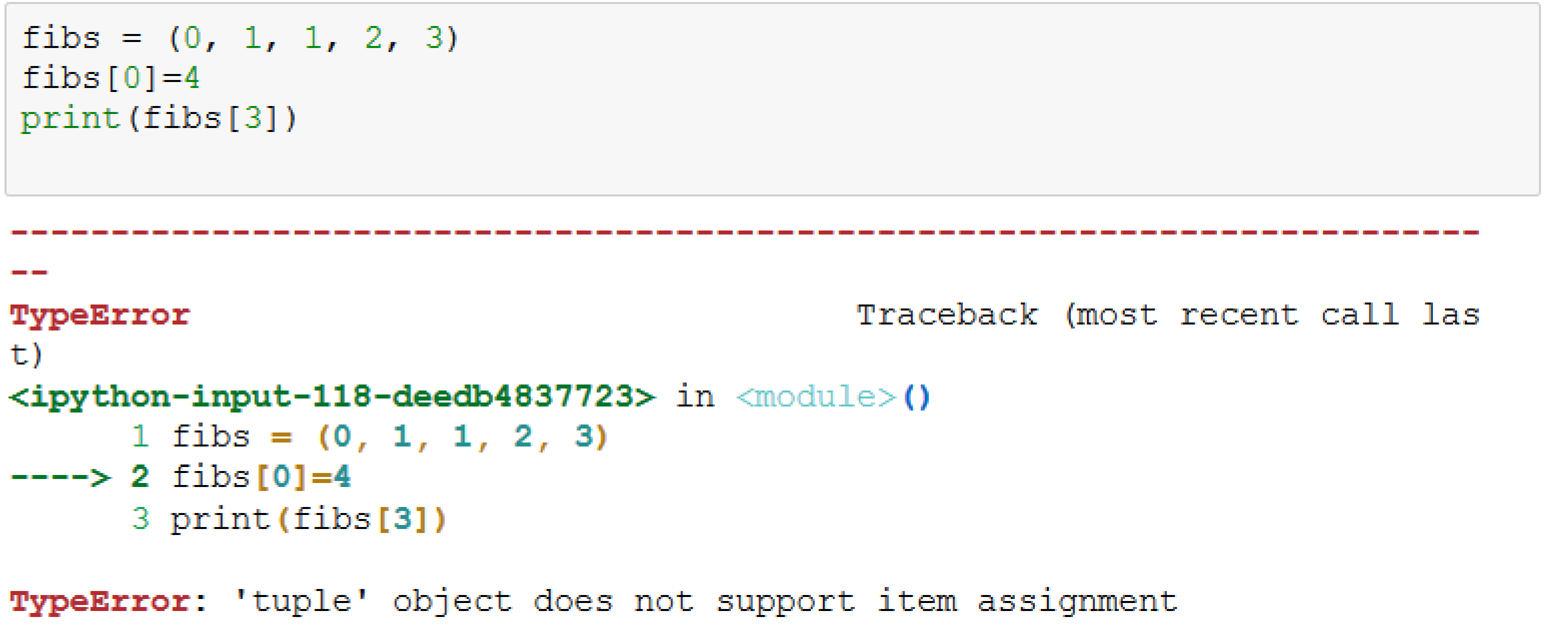
A tuple is like a list that uses parentheses, as in this example:

```
fibs = (0, 1, 1, 2, 3)
print(fibs[3])
```

```
2
```

Here we define the variable fibs as the numbers 0, 1, 1, 2, and 3. Then, as with a list, we print the item in index position 3 in the tuple using print(fibs[3]). The main difference between a tuple and a list is that a tuple cannot change once you've created it.

For example, if we try to replace the first value in the tuple fibs with the number 4 (just as we replaced values in our wizard_list), we get an error message:



```
fibs = (0, 1, 1, 2, 3)
fibs[0]=4
print(fibs[3])
```

```
---------------------------------------------------------------------------
--
TypeError                                 Traceback (most recent call las
t)
<ipython-input-118-deedb4837723> in <module>()
      1 fibs = (0, 1, 1, 2, 3)
----> 2 fibs[0]=4
      3 print(fibs[3])

TypeError: 'tuple' object does not support item assignment
```

So why use tuple?

Why would you use a tuple instead of a list? Basically because sometimes it is useful to use something that you know can never change. If you create a tuple with two elements inside, it will always have those two elements inside.

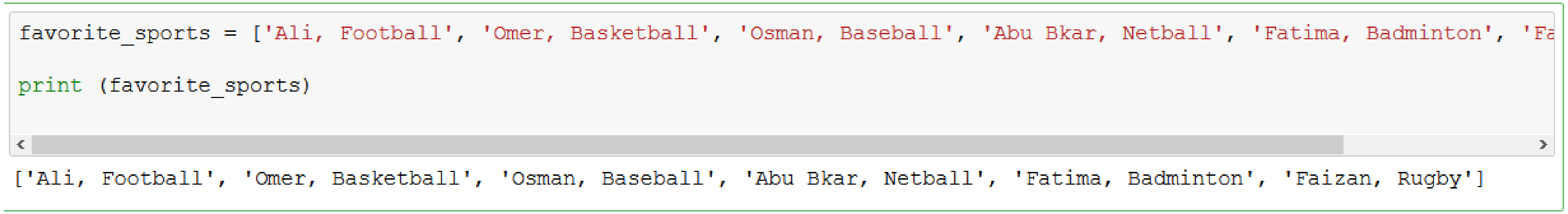
Python Maps

In Python, a map (also referred to as a dict, short for dictionary) is a collection of things, like lists and tuples. The difference between maps and lists or tuples is that each item in a map has a key and a corresponding value. For example, say we have a list of people and their favorite sports. We could put this information into a Python list, with the person’s name followed by their sport, like so:

Creating a map in Python

favorite_sports = ['Ali, Football', 'Omer, Basketball', 'Osman, Baseball', 'Abu Bkar, Netball', 'Fatima, Badminton', 'Faizan, Rugby']

print (favorite_sports)



```
favorite_sports = ['Ali, Football', 'Omer, Basketball', 'Osman, Baseball', 'Abu Bkar, Netball', 'Fatima, Badminton', 'Fa

print (favorite_sports)
```

```
['Ali, Football', 'Omer, Basketball', 'Osman, Baseball', 'Abu Bkar, Netball', 'Fatima, Badminton', 'Faizan, Rugby']
```

Modifying a value in a map.

Now, if we store this same information in a map, with the person’s name as the key and their favorite sport as the value, the Python code would look like this:

```
favorite_sports = {'Ali' : 'Football',
                'Omer' : 'Basketball',
                'Osman' : 'Baseball',
                'Abu Bakr' : 'Netball',
                'Fatima' : 'Badminton',
                'Faizan' : 'Rugby'}

print (favorite_sports)
```

We use colons to separate each key from its value, and each key and value is surrounded by single quotes. Notice, too, that the items in a map are enclosed in braces ({}), not parentheses or square brackets.

Now to get Ali favourite sport, we access ou rmap favourite_sports using his name as

```
favorite_sports = {'Ali' : 'Football',
               'Omer' : 'Basketball',
               'Osman' : 'Baseball',
               'Abu Bakr' : 'Netball',
               'Fatima' : 'Badminton',
               'Faizan' : 'Rugby'}

print(favorite_sports['Ali'])
```

Deleting a value in the map

```
favorite_sports = {'Ali' : 'Football',
                   'Omer' : 'Basketball',
                   'Osman' : 'Baseball',
                   'Abu Bakr' : 'Netball',
                   'Fatima' : 'Badminton',
                   'Faizan' : 'Rugby'}

del favorite_sports['Ali']

print(favorite_sports)
```

Replacing a value in a map

```
favorite_sports = {'Ali' : 'Football',
                   'Omer' : 'Basketball',
                   'Osman' : 'Baseball',
                   'Abu Bakr' : 'Netball',
                   'Fatima' : 'Badminton',
                   'Faizan' : 'Rugby'}

favorite_sports['Faizan'] = 'Cricket'

print(favorite_sports)
```

We replace the favorite sport of Rugby with Cricket by using the key Faizan. As you can see, working with maps is kind of like working with lists and tuples, except that you can’t join maps with the plus operator (+). If you try to do that, you’ll get an error message:

جزاك اللهُ

Please send us your questions at zone@alnafi.com
We will only answer our Nafi Members. So please quote your membership number within the email.