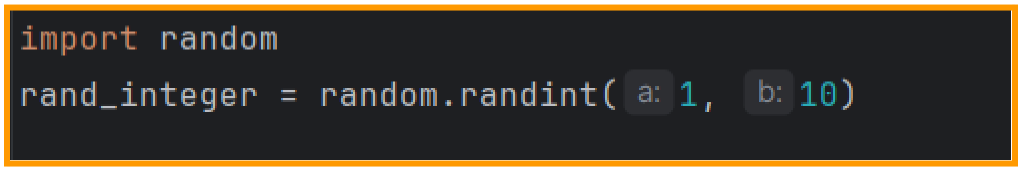
Day 4: Randomisation and Python lists

Randomization is really, really important when we want to create computer programs that have a degree of unpredictability. Now the biggest category of that is of course, games. We've talked a lot about randomness, but let's see it in action. The python team already created a random module.
https://docs.python.org/3/library/random.html You can find here a whole bunch of functions that you can use that allows you to generate random integers. So random whole numbers, or random floating point numbers. In this case were going to use:
`random.randint(a, b)`

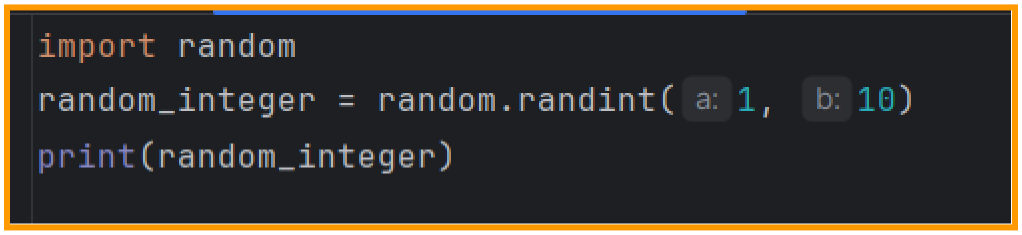
Return a random integer *N* such that `a <= N <= b`. Alias for `randrange(a, b+1)`.

Allow us to choose a random number between a and b, which can be 10 and 20.



```python
import random
rand_integer = random.randint( a: 1,  b: 10)
```

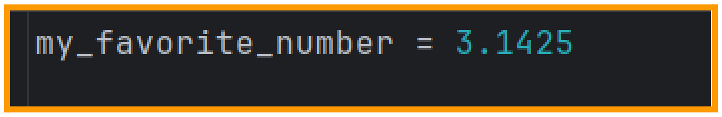
So in this case I've got 1 as the first starting value and then it goes up to 10 and includes 10. So it means that we will get random numbers between 1 and 10.



```python
import random
random_integer = random.randint( a: 1,  b: 10)
print(random_integer)
```

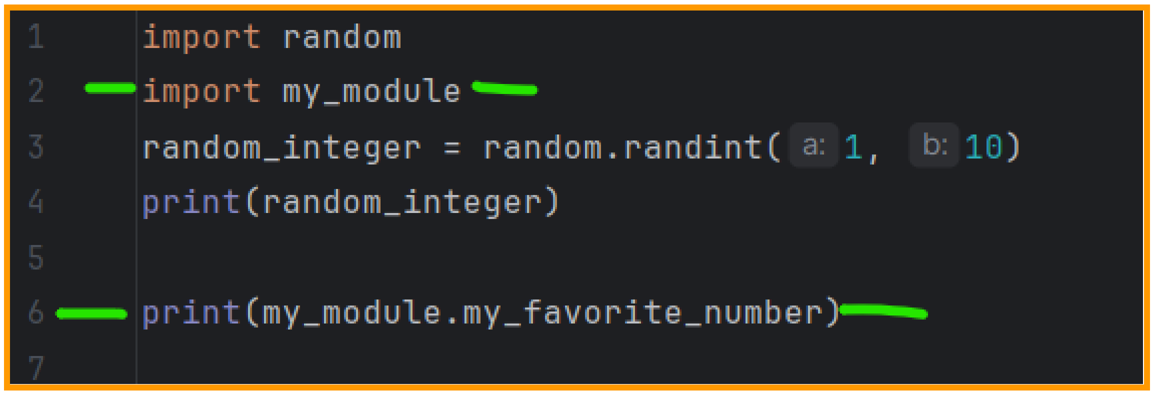
Lets print it and every single time we run it we will get a random number.Now, I've mentioned that the Random Module is a Python module. So what exactly is a module? Well, you've seen that we've mostly been writing our code all on the same page in a sort of script and everything just kind of gets executed from top to bottom. Sometimes your code will get so long that it will be kinda complicated to understand whats going on, in this case what often people do is split the code into individual modules and sometimes you would have different people working in different modules.

Now we're going to create our Python file and call it my_module, so that I can show you how to create your very own module. Random module<new<Python file< My_module
Then you create a variable:



```python
my_favorite_number = 3.1425
```

Now, if I want to use this number that I've created in my_module, all I have to do is to go back into the task.py, which is the main entry point, at least in this course format.
Now if I want to use my code from my module, the only thing I have to do is import same as the random number:

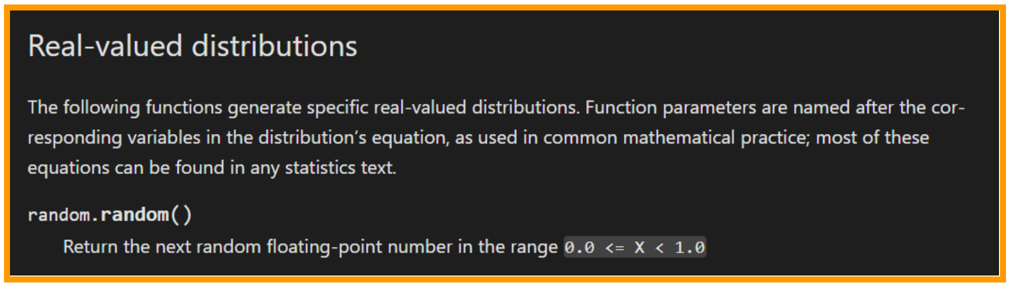


```python
import random
import my_module
random_integer = random.randint( a: 1,  b: 10)
print(random_integer)

print(my_module.my_favorite_number)
```

And now if I go ahead and print, you can see that my_favorite_number is printed by using and tapping into this module.

Generating floating point numbers

Well, let's take a look at the documentation. If you scroll down to this area called Real-valued distributions, you can see that one way of generating random floating point numbers is using the random.random() function.

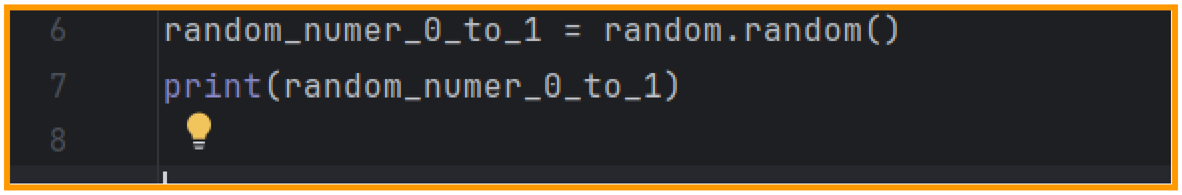


Real-valued distributions

The following functions generate specific real-valued distributions. Function parameters are named after the corresponding variables in the distribution's equation, as used in common mathematical practice; most of these equations can be found in any statistics text.

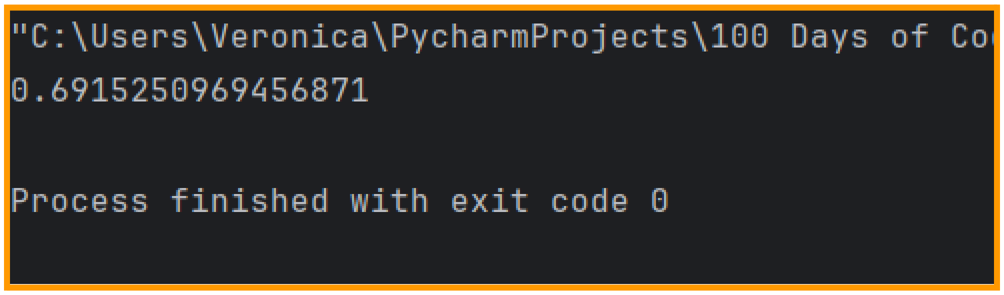
`random.random()`
Return the next random floating-point number in the range `0.0 <= X < 1.0`

And this is one of the core functions that we use a lot when we're dealing with random numbers in Python, because it is one of the most adaptive functions. Now if you look at these symbols, you'll see that the random number that's generated can be greater or equal to zero. So that means zero is included in the possibility, but it is less than one. So one is not in the generated random numbers.



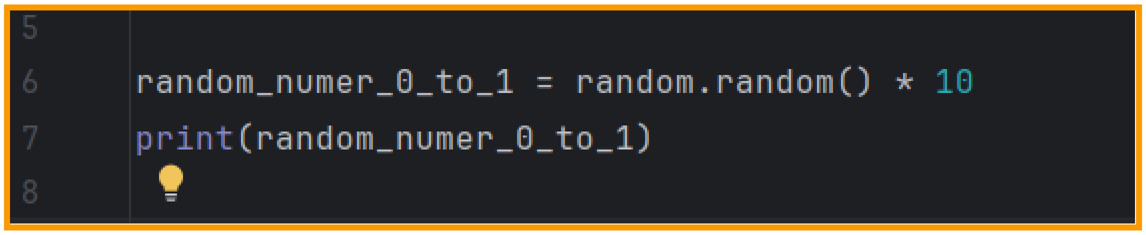
```python
random_numer_0_to_1 = random.random()
print(random_numer_0_to_1)
```

every time I press the print button, it generates a new floating point number, and it's between 0 inclusive and up to 1, not inclusive.



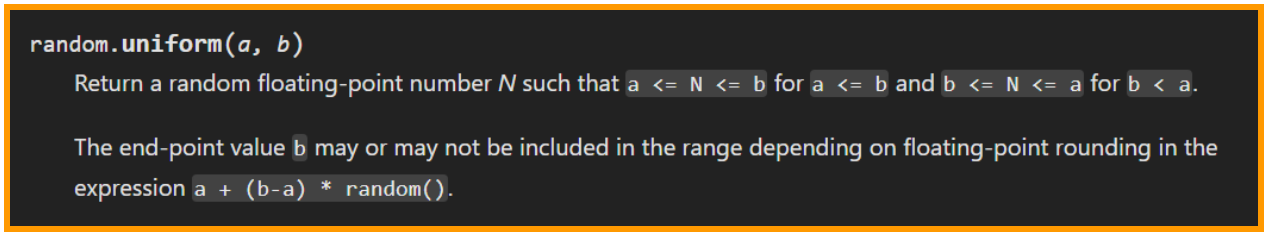
```
"C:\Users\Veronica\PycharmProjects\100 Days of Co
0.6915250969456871

Process finished with exit code 0
```

So you can use this random number generator, random.random() to expand it. If we multiply this by 10, then you'll see that the numbers that are generated are going to be between 0 to 10 and not inclusive of 10, because we're basically multiplying the lower range 0 by 10, which is 0, and the upper range by 10, 1 * 10 is 10.



```python
random_numer_0_to_1 = random.random() * 10
print(random_numer_0_to_1)
```

So then we've shifted this random number generator to create floating point numbers between 0 and whatever it is that we desire. I you look the documentation youre going to discover another function that is called Random Floating Point Number Generator

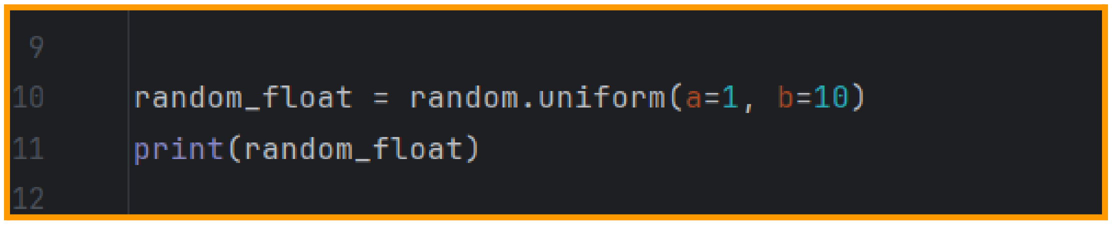


```
random.uniform(a, b)
```

Return a random floating-point number *N* such that `a <= N <= b` for `a <= b` and `b <= N <= a` for `b < a`.

The end-point value `b` may or may not be included in the range depending on floating-point rounding in the expression `a + (b-a) * random()`.

So basically it's going to generate numbers between a and b, inclusive of a and b.

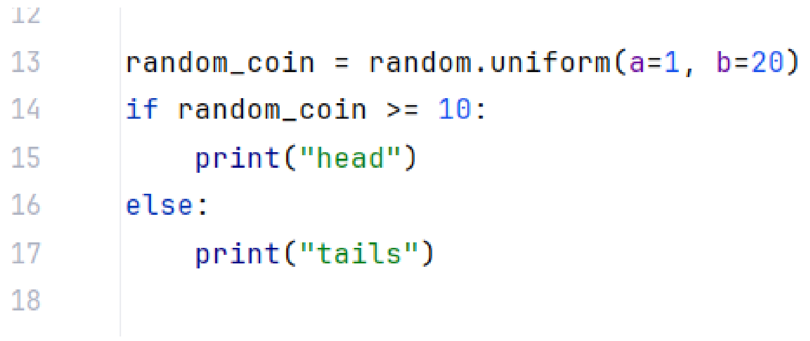


```python
random_float = random.uniform(a=1, b=10)
print(random_float)
```

And now when I run this you can see it almost does the same thing. The only difference is it's not entirely sure depending on rounding, whether you will get 10 or not in the random_float.

Exercise

Create a coin flip program using what you have learnt about randomisation in Python. It should randomly print "Heads" or "Tails" everytime it is run.

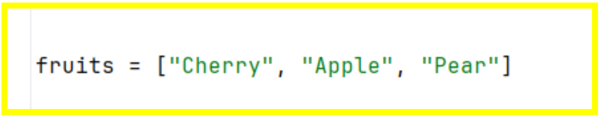


```python
random_coin = random.uniform(a=1, b=20)
if random_coin >= 10:
    print("head")
else:
    print("tails")
```

I did it like this but you can use the values that you want

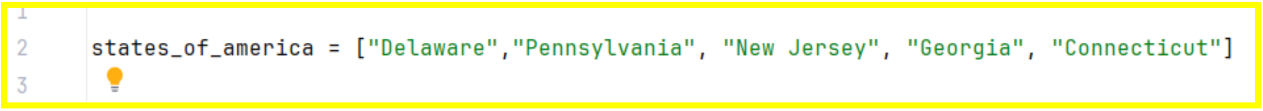
Understanding the Offset and Appending Items to Lists

Python list is what you would call Data Structure, it means well, it's just a way of organizing and storing data in Python. We've seen how to store single pieces of data on variables But sometimes you might want to store grouped pieces of data, data that has some sort of connection with each other. Now, in other cases, you might also want to have order in your data. And lists look pretty simple, it's just a set of square brackets with many items stored inside, and those items can be any data type. It doesn't really matter, but what does matter is the syntax. In Python, lists always start with an open square bracket like this - [, and a closing square bracket like this -].
If we stored a bunch of fruits, for example, then it might look something like this:



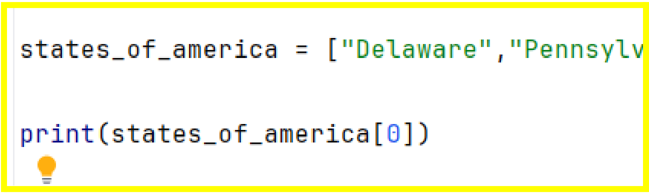
```
fruits = ["Cherry", "Apple", "Pear"]
```

Let's take a look using real code, let's say that I wanted to store all of the names of the states of the US.

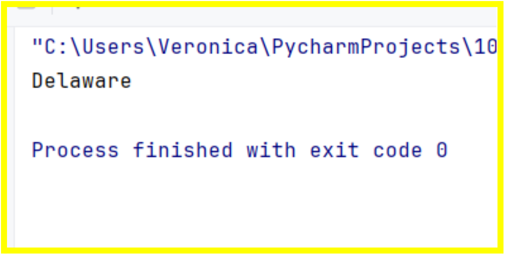


```
states_of_america = ["Delaware","Pennsylvania", "New Jersey", "Georgia", "Connecticut"]
```

The order is determined by the order in the list and when you store it inside the variable, that order is not lost and you'll be able to use it later on when you need the list. if later on I decided that I wanted to know which was the state that joined first, then I can print this variable states_of_america, I can add a set of square brackets, and then I type 0 as the index of the piece of data I want to pull out from my states_of_america list.

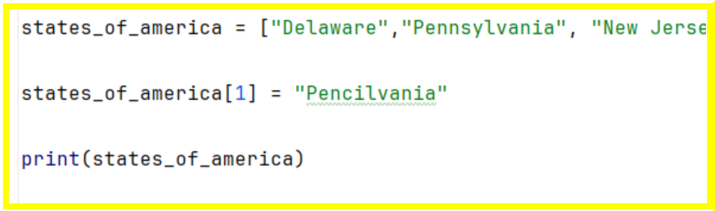


```
states_of_america = ["Delaware","Pennsylv

print(states_of_america[0])
```

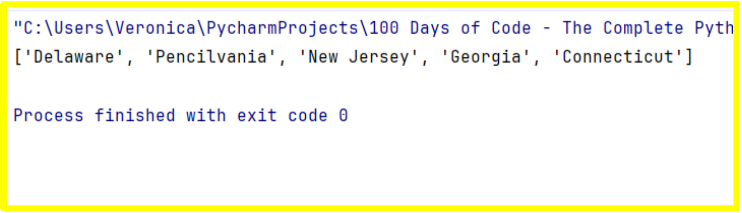


```
"C:\Users\Veronica\PycharmProjects\10
Delaware

Process finished with exit code 0
```

Remember that programmers start counting from 0
So whenever you see square brackets, you should be thinking to yourself, oh, this might be related to a list, because when you create the list, you use square brackets, and when you try to get items out of the list, you also use square brackets. Remember you can use negative numbers to count So if I wrote -1 or -2, then it actually starts counting from the end of the list. You can also change the items in the list using very similar code. For example, if I decided that Pennsylvania is actually not spelled Pennsylvania and I wanted to change

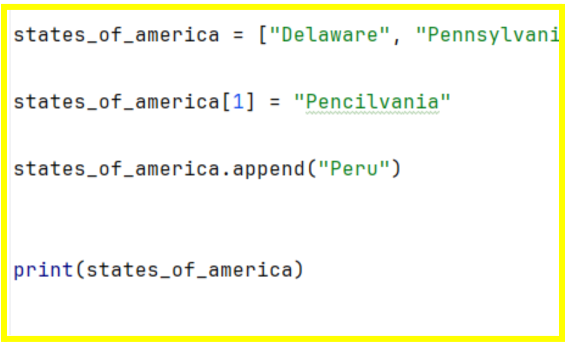


```
states_of_america = ["Delaware","Pennsylvania", "New Jerse

states_of_america[1] = "Pencilvania"

print(states_of_america)
```

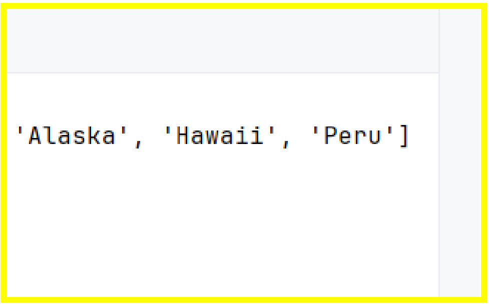


```
"C:\Users\Veronica\PycharmProjects\100 Days of Code - The Complete Pyth
['Delaware', 'Pencilvania', 'New Jersey', 'Georgia', 'Connecticut']

Process finished with exit code 0
```

Instead of Pennsylvania, it's now Pencilvania. So you can alter any item inside the list pretty easily using this kind of syntax. You can also add to the list if you want to. Let's add something at the end of the list.Let's say that Peru is joining the United States of America.

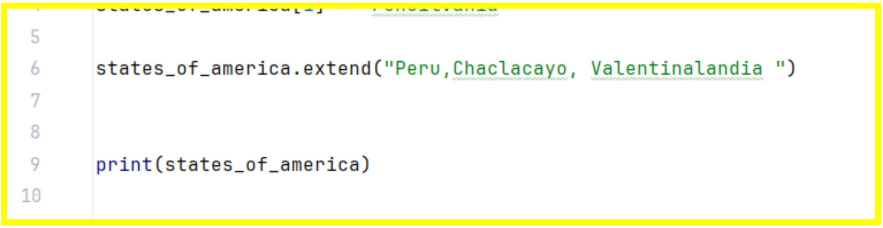


```
states_of_america = ["Delaware", "Pennsylvani

states_of_america[1] = "Pencilvania"

states_of_america.append("Peru")


print(states_of_america)
```



```
'Alaska', 'Hawaii', 'Peru']
```

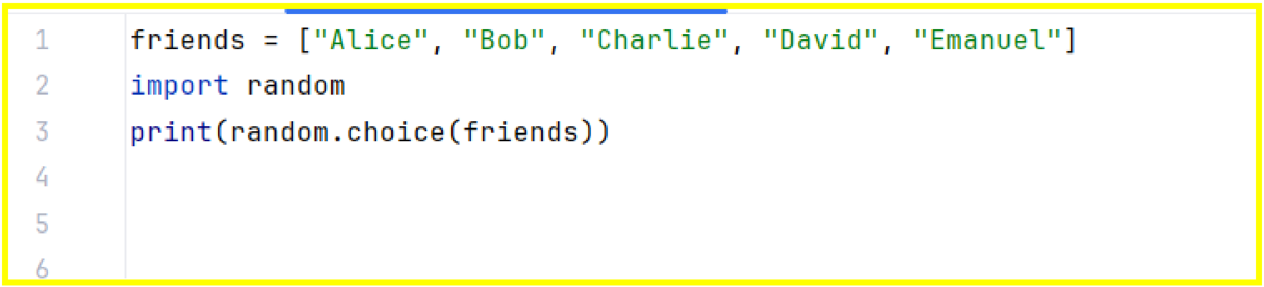
append() will add a single item to the end of the list. Now, there's actually a whole load of other functions that you can use in addition to append(), and you'll find this on the documentation for Python: https://docs.python.org/3/tutorial/datastructures.html
For example, you can use the extend() which adds a whole bunch of items to the end of the list.



```

states_of_america.extend("Peru,Chaclacayo, Valentinalandia ")


print(states_of_america)

```

Code Challenge: Who will pay the bill?

So here's the challenge, you are to look through this friends list and you're going to write some code using what you've learned about randomization in Python, and also lists in Python to be able to print out a random, one of these names.

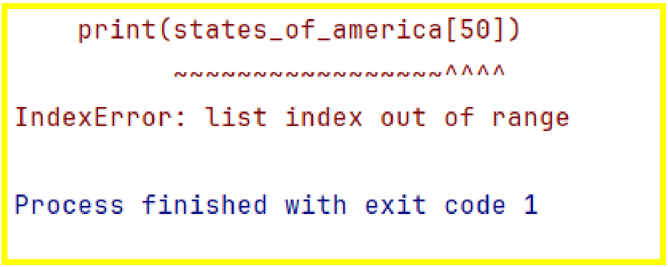


```python
friends = ["Alice", "Bob", "Charlie", "David", "Emanuel"]
import random
print(random.choice(friends))
```

Read the documentation.

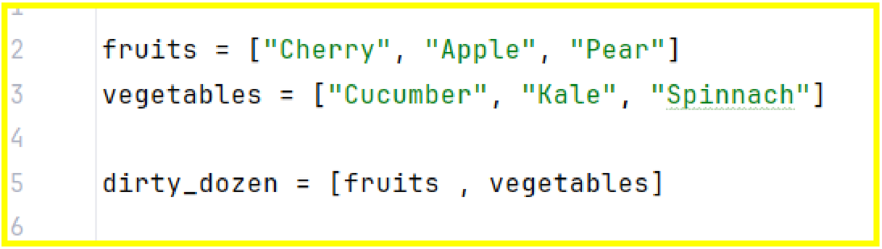
IndexErrors and Working with Nested Lists

One of the most common errors,called the index out of range error. What happens if we put between brackets 50 wanting to get Hawaii, well get this.

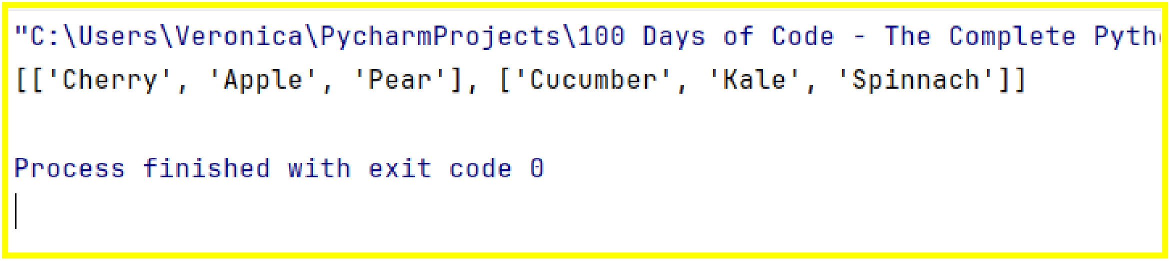


```
    print(states_of_america[50])
          ~~~~~~~~~~~~~~~~~^^^^
IndexError: list index out of range

Process finished with exit code 1
```

But when you're working with large lists and you're not always looking at the data, then these errors can be a little bit more confusing. Very frequently when you're working with lists, you'll end up with an off-by-one error. Lets create an example with a dirty dozen (12 fruits and vegetables that have the most pesticide) But some of them are fruits and the other ones are pesticides So how can we use our lists to still keep them inside the same sort of container? We could create two separate lists but they kind of have a relationship between them So how can we have lists within a list? Well, that's what's called a nested list. Instead of our original dirty_dozen, we could create a new list called dirty_dozen, and we set it equal to a list that contains two lists. It contains fruits and it contains vegetables.


```
fruits = ["Cherry", "Apple", "Pear"]
vegetables = ["Cucumber", "Kale", "Spinnach"]

dirty_dozen = [fruits , vegetables]
```

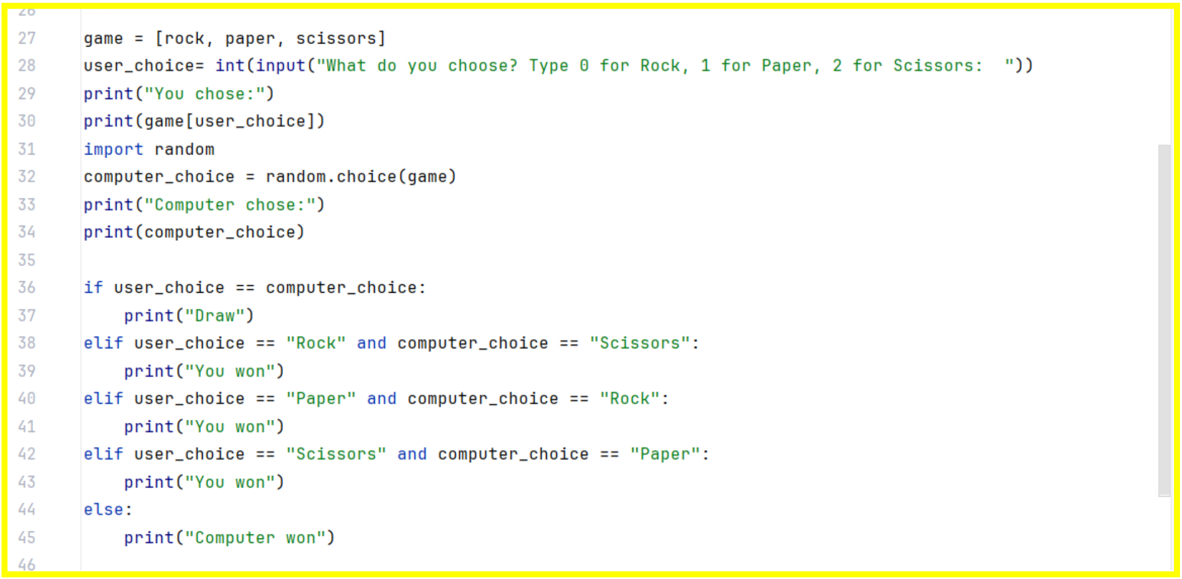
And if I go ahead and print out this list you'll be able to see its structure. Looks like this:


```
"C:\Users\Veronica\PycharmProjects\100 Days of Code - The Complete Pyth
[['Cherry', 'Apple', 'Pear'], ['Cucumber', 'Kale', 'Spinnach']]

Process finished with exit code 0
```

You'll notice that there's two brackets at the beginning and at the end and its because its two separate list.

Project Rock Paper Scissors:

You are going to build a Rock, Paper, Scissors game. You will need to use what you have learnt about randomisation and Lists to achieve this.


```
game = [rock, paper, scissors]
user_choice= int(input("What do you choose? Type 0 for Rock, 1 for Paper, 2 for Scissors:  "))
print("You chose:")
print(game[user_choice])
import random
computer_choice = random.choice(game)
print("Computer chose:")
print(computer_choice)

if user_choice == computer_choice:
    print("Draw")
elif user_choice == "Rock" and computer_choice == "Scissors":
    print("You won")
elif user_choice == "Paper" and computer_choice == "Rock":
    print("You won")
elif user_choice == "Scissors" and computer_choice == "Paper":
    print("You won")
else:
    print("Computer won")
```

Notes:
Look at the int(
The and, or