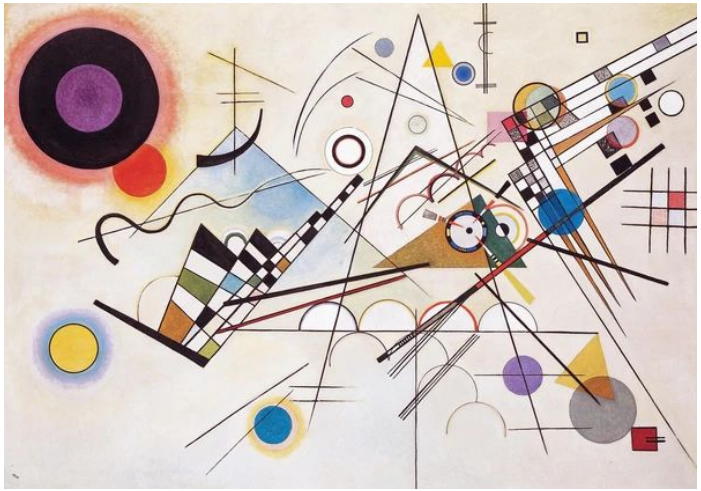


JamCoders: Week 1

Lecture 5A:

- Nested Lists
- Nested Loops
- Modules & Imports

Nested Lists

2-Dimensional Lists

We learned about lists as an ordered collection of items of any type.

Creating a List

A **list** is an ordered collection of items of any type.

Create a new list using brackets, with a comma-separated list of values.

```
players = ["Ronaldo", "Messi", "Drogba"]
```

0	1	2
"Ronaldo"	"Messi"	"Drogba"

2-Dimensional Lists

We learned about lists as an ordered collection of items of <u>any type</u>.

Creating a List

A **list** is an ordered collection of items of any type.

Create a new list using brackets, with a comma-separated list of values.

```
players = ["Ronaldo", "Messi", "Drogba"]
```

0	1	2
"Ronaldo"	"Messi"	"Drogba"

2-Dimensional Lists

When you have a list that holds *other lists*, we call this a “2 dimensional list”

2-Dimensional Lists

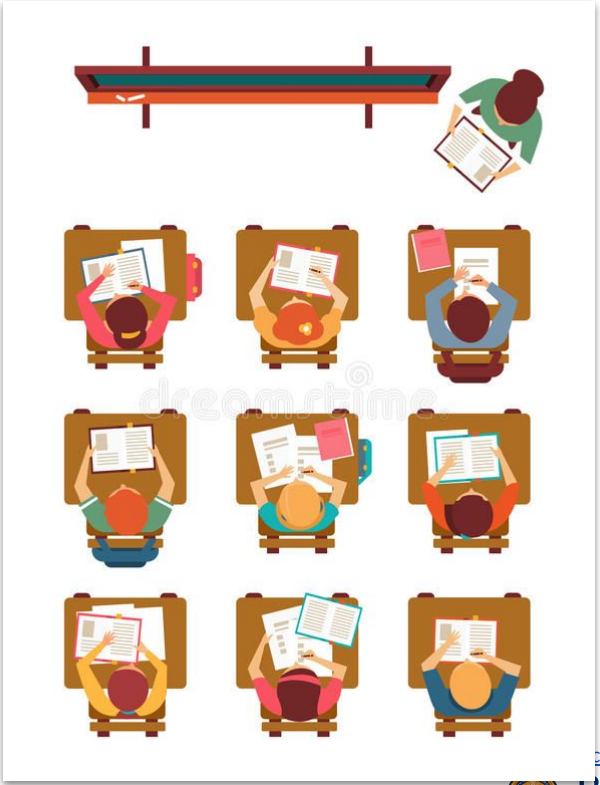
When you have a list that holds *other lists*, we call this a “2 dimensional list”

```
seating = [
  ['Anton', 'Jordan', 'Joy'],
  ['Ibrahim', 'Sam', 'Pooja'],
  ['Okoro', 'Nathan', 'Mira']
]
```

2-Dimensional Lists

It’s useful to think of them as representing some sort of grid.

```
seating = [
  ['Anton', 'Jordan', 'Joy'],
  ['Ibrahim', 'Sam', 'Pooja'],
  ['Okoro', 'Nathan', 'Mira']
]
```

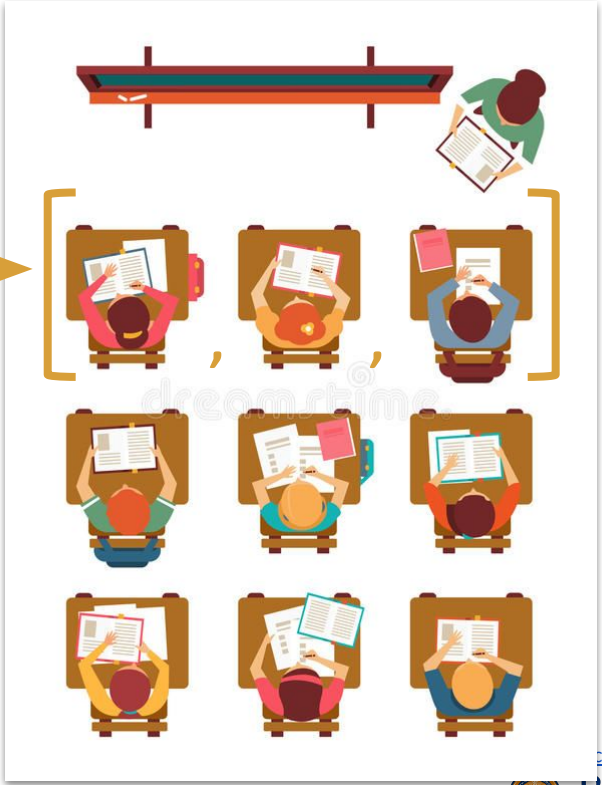


2-Dimensional Lists

It's useful to think of them as representing some sort of grid.

```
seating = [
  ['Anton', 'Jordan', 'Joy'],
  ['Ibrahim', 'Sam', 'Pooja'],
  ['Okoro', 'Nathan', 'Mira']
]
print(seating[0])
```

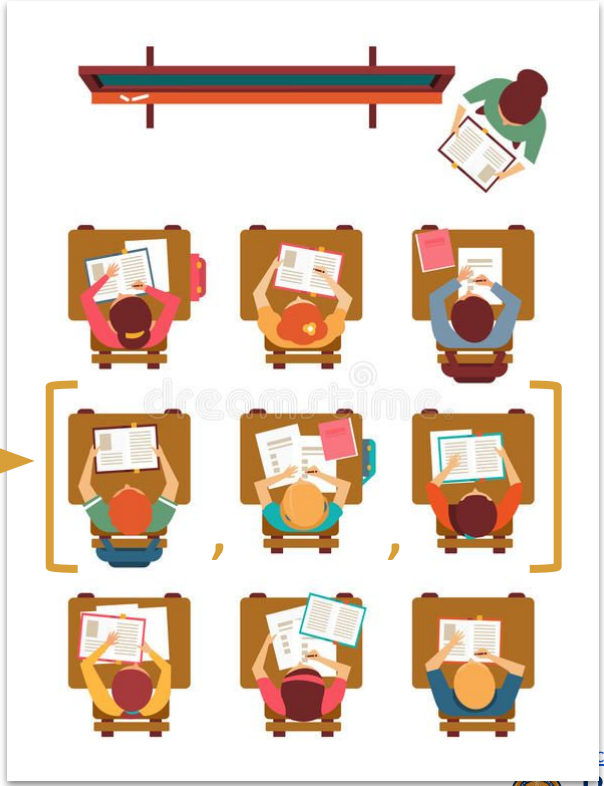
```
> ['Anton', 'Jordan', 'Joy']
```



2-Dimensional Lists

It’s useful to think of them as representing some sort of grid.

```
seating = [
  ['Anton', 'Jordan', 'Joy'],
  ['Ibrahim', 'Sam', 'Pooja'],
  ['Okoro', 'Nathan', 'Mira']
]
print(seating[1])
```



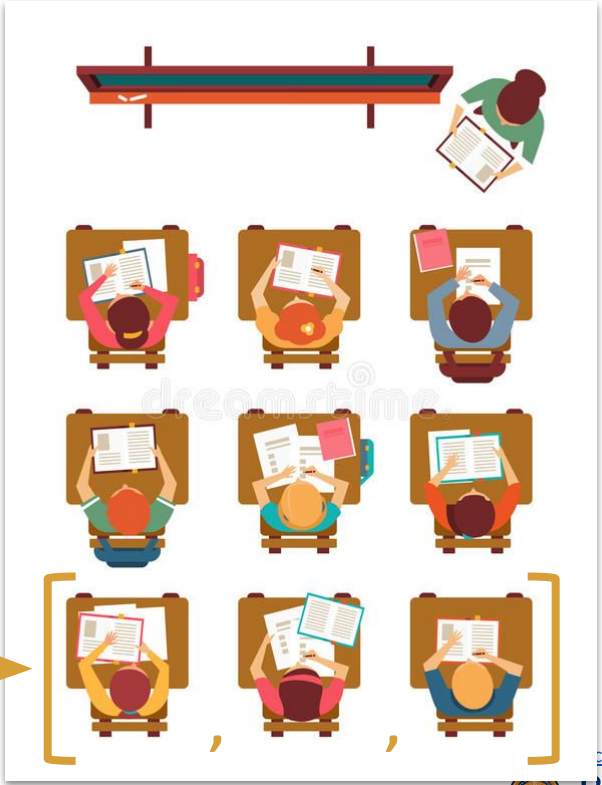
```
> ['Ibrahim', 'Sam', 'Pooja']
```

2-Dimensional Lists

It's useful to think of them as representing some sort of grid.

```
seating = [
  ['Anton', 'Jordan', 'Joy'],
  ['Ibrahim', 'Sam', 'Pooja'],
  ['Okoro', 'Nathan', 'Mira']
]
print(seating[2])
```

```
> ['Okoro', 'Nathan', 'Mira']
```

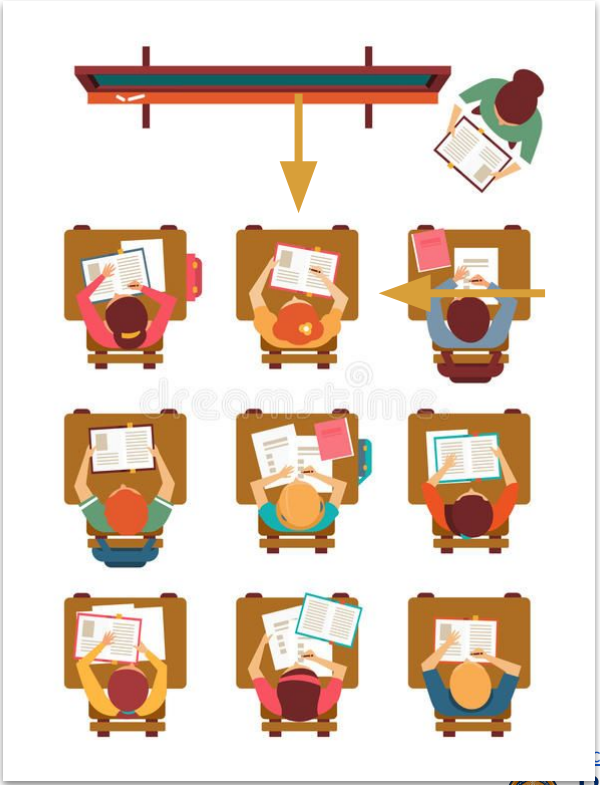


2-Dimensional Lists

To get a particular item in an inner list, we first index into the list to get the correct “row”, then index into that “row” to get the item we want.

```
seating = [
  ['Anton', 'Jordan', 'Joy'],
  ['Ibrahim', 'Sam', 'Pooja'],
  ['Okoro', 'Nathan', 'Mira']
]
print(seating[0][1])
```

> **Jordan**



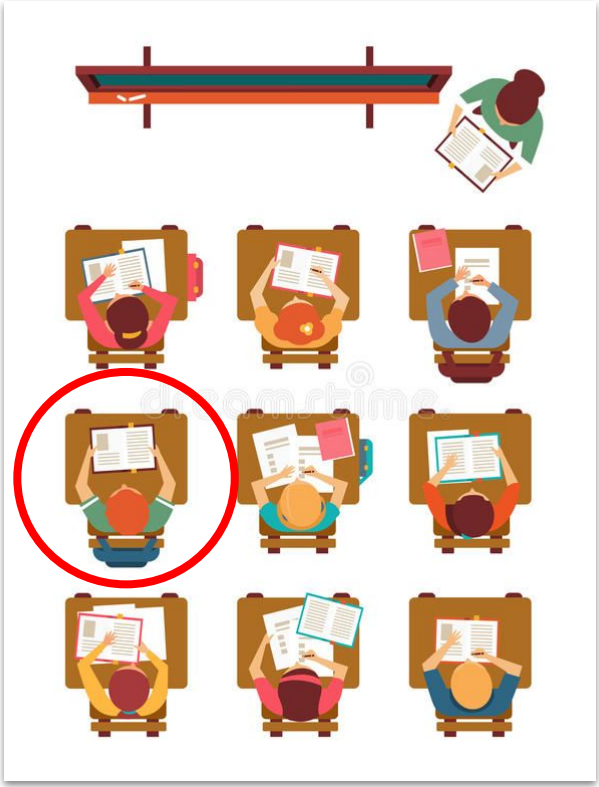
2-Dimensional Lists

Indexing into a list is an *expression*. So the outer index is applied to the value that the inner index returns.

```
seating[0][1]
```

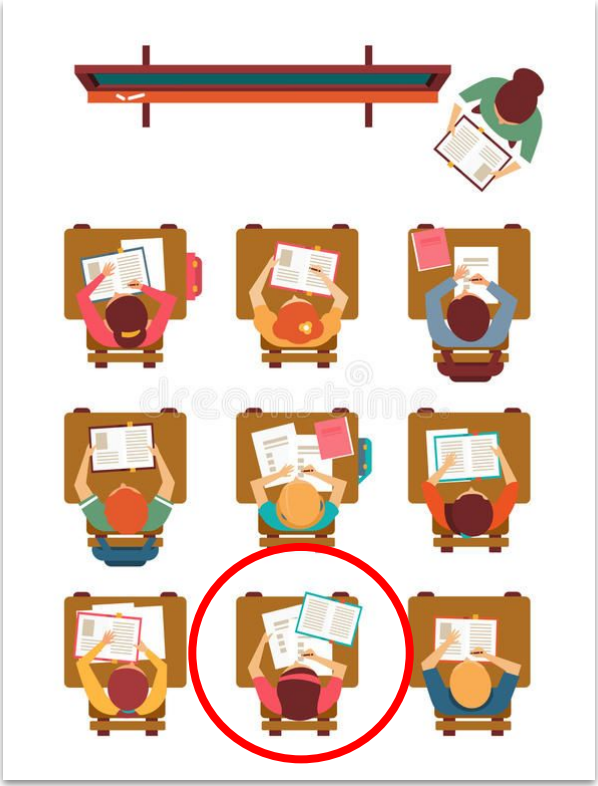
2-Dimensional Lists

```
seating[1][0]
```



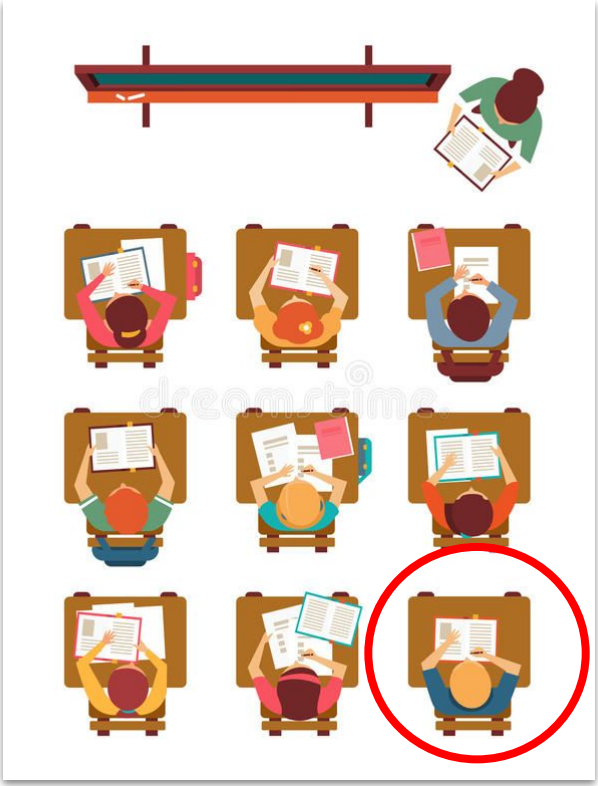
2-Dimensional Lists

```
seating[2][1]
```



2-Dimensional Lists

```
seating[2][2]
```



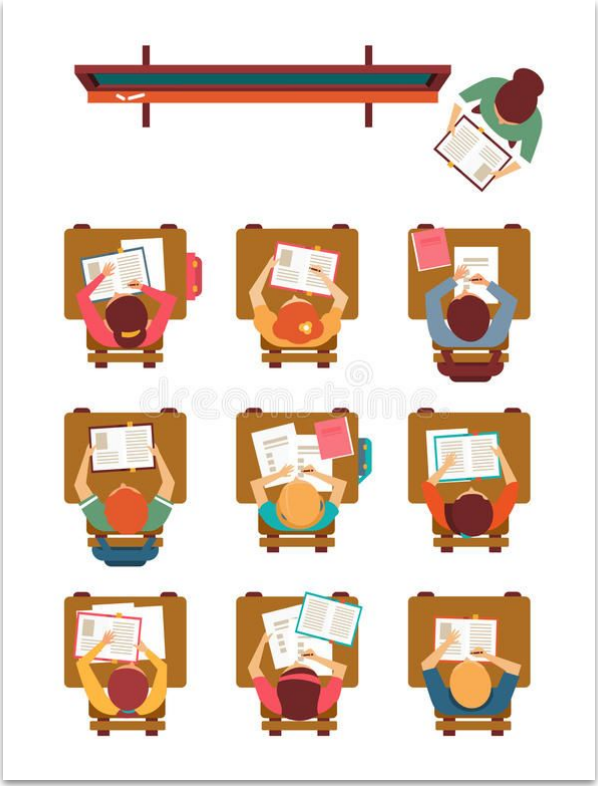
Nested Loops

2D List Iterating

What if we want iterate over all of the seats?

```
seating = [
  ['Anton', 'Jordan', 'Joy'],
  ['Ibrahim', 'Sam', 'Pooja'],
  ['Okoro', 'Nathan', 'Mira']
]
????
```

> Anton
> Jordan
> Joy
> Ibrahim
> Sam
> Pooja
> Okoro
> Nathan
> Mira



Nested For Loops

For each row, we will *for each name in that row* print it to the screen.

Nested For Loops

For each row, we will *for each name in that row* print it to the screen.

```
???
```

Nested For Loops

For each row, we will *for each name in that row* print it to the screen.

```
# For each row index
for row in range(3):
    # For this particular row,
    # print each entry
    ???
```

Nested For Loops

For each row, we will *for each name in that row* print it to the screen.

```
# For each row index
for row in range(3):
    # For each column index
    for col in range(3):
        print(seats[row][col])
```

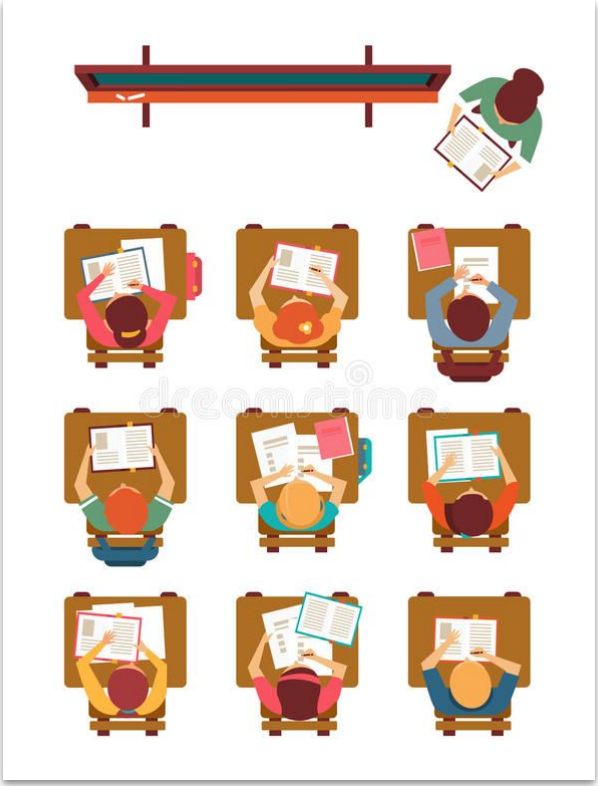
See it in action on PythonTutor

2D List Iterating

The *other* way to see this…

This is how we get all of the names:

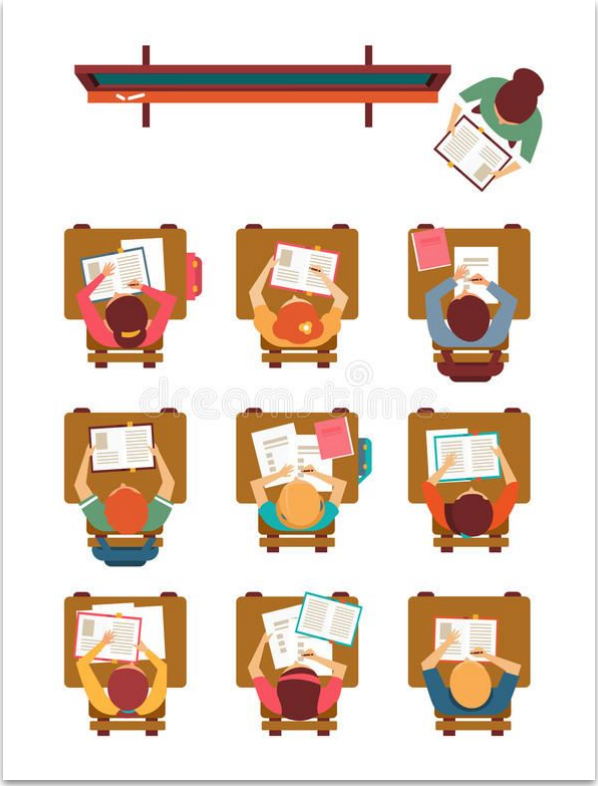
seating[0][0]
seating[0][1]
seating[0][2]
seating[1][0]
seating[1][1]
seating[1][2]
seating[2][0]
seating[2][1]
seating[2][2]



2D List Iterating

Notice the pattern. Repeating [0,1,2] outside

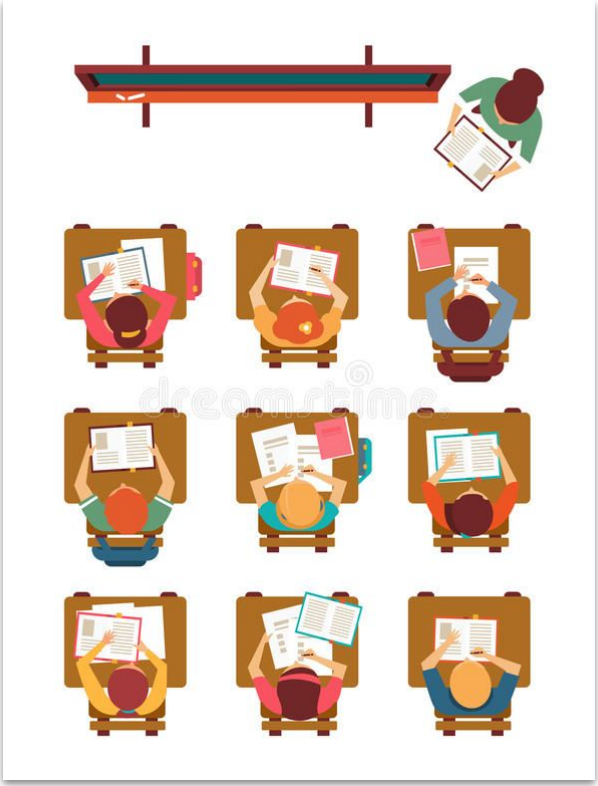
seating[0][0]
seating[0][1]
seating[0][2]
seating[1][0]
seating[1][1]
seating[1][2]
seating[2][0]
seating[2][1]
seating[2][2]



2D List Iterating

Notice the pattern. Each repeated [0,1,2] is matched up with one digit (the row)

```
seating[0][0]
seating[0][1]
seating[0][2]
seating[1][0]
seating[1][1]
seating[1][2]
seating[2][0]
seating[2][1]
seating[2][2]
```



2D List Iterating

Notice the pattern. Each repeated [0,1,2] is matched up with one digit (the row)

seating[0][0]
seating[0][1]
seating[0][2]
seating[1][0]
seating[1][1]
seating[1][2]
seating[2][0]
seating[2][1]
seating[2][2]

```
# For each row index
for row in range(3):
    # For each column index
    for col in range(3):
        print(row, col)
```

This is the code that would generate all of these pairs of numbers together.

Modules & Imports

Modules and Imports

Just like we use *functions* to organize our code, we can use **modules** to organize larger codebases and import code that others have written.

YouTube is not written all in one big file called youtube.py. Different functionality is split across different *modules*, which different people own.

Modules and Imports

module: a file containing Python definitions and statements, written and shared by another programmer.

import statement: loads module into program to make its functions available

Modules and Imports

A “module” is just a **Python file containing definitions and statements**. You can make your own!

If it’s in the same folder (file.py), import it with `**import file**`, then use functions with `file.function_name()`

main.py

```
import helper

val = helper.max_val(15, 12)
helper.pretty_print("Value is " + str(val))
```

helper.py

```
def pretty_print(string):
  print("-----------------")
  print(string)
  print("-----------------")

def max_val(a, b):
    if a > b:
        return a
    return b
```

Modules and Imports

Let’s learn our first module: **random**

Modules and Imports

Let’s learn our first module: **random**

```
import random

rand_int = random.randrange(0, 10)
print(rand_int)
```

Modules and Imports

Let's learn our first module: **random**

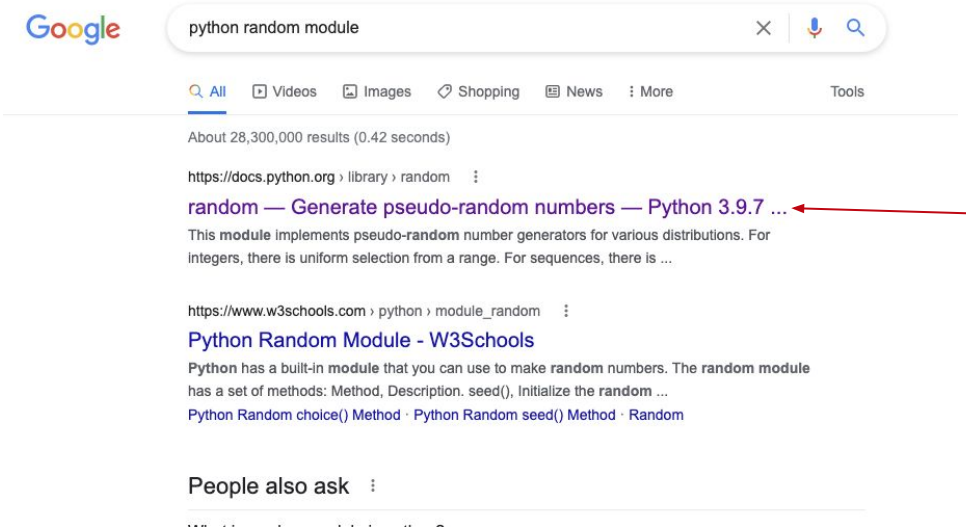
```
import random

rand_int = random.randrange(0, 10)
print(rand_int)
```

How did I know this?

The **randrange(start, end)** function of the **random** module a random integer from the range [start, end) where end is exclusive (just like regular range)

Modules and Imports

Documentation: description of the functions a programmer writes.



https://docs.python.org/3/library/random.html

Random Module

Important skill: learn to find and read documentation online.

When you don't know how to do something, *search for the documentation*.

Functions for integers

random.**randrange**(*stop*)

random.**randrange**(*start*, *stop*[, *step*])

Return a randomly selected element from `range(start, stop, step)`. This is equivalent to `choice(range(start, stop, step))`, but doesn't actually build a range object.

The positional argument pattern matches that of `range()`. Keyword arguments should not be used because the function may use them in unexpected ways.

Changed in version 3.2: `randrange()` is more sophisticated about producing equally distributed values. Formerly it used a style like `int(random()*n)` which could produce slightly uneven distributions.

random.**randint**(*a*, *b*)

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

random.**getrandbits**(*k*)

Returns a non-negative Python integer with *k* random bits. This method is supplied with the MersenneTwister generator and some other generators may also provide it as an optional part of the API. When available, `getrandbits()` enables `randrange()` to handle arbitrarily large ranges.

Changed in version 3.9: This method now accepts zero for *k*.