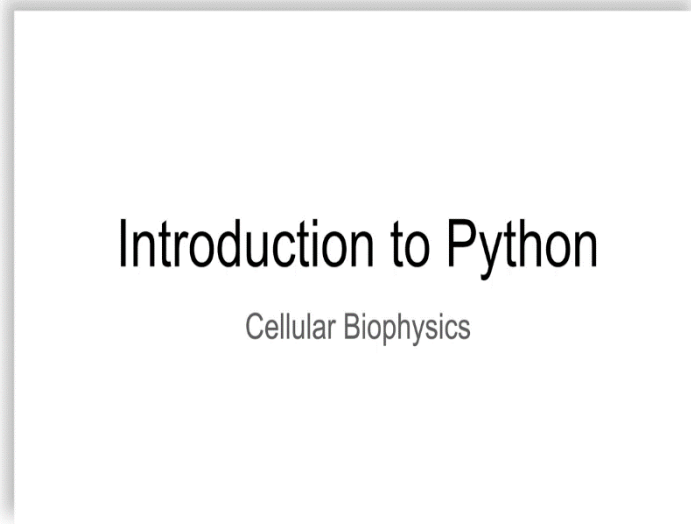


Cellular Biophysics
Doctor Chaitanya Athale.
Department of Biology
Indian Institute of Science Education and Research, Pune
Lecture 02
Python Programming

(Refer Slide Time: 00:16)



I had, in the previous module, talked a little bit about a very brief introduction to Python. And for those who have not been programming before, I would not worry too much, because in this principle of learn as you go, LAGO, like I like to call it, easy as a play game, we are going to pick up concepts as we go along.

In other words, so long as you can run very basic lines of code, the rest we are going to pick up on the way, and that is part of what I want you to learn. Now, many of you are more advanced programmers, and for those of you can easily skip this video a little ahead if you like. The conclusion of this part however is an assignment, so if you already know everything, clearly do the assignment, and you are done. So, what is the assignment?

(Refer Slide Time: 01:08)

Assignment

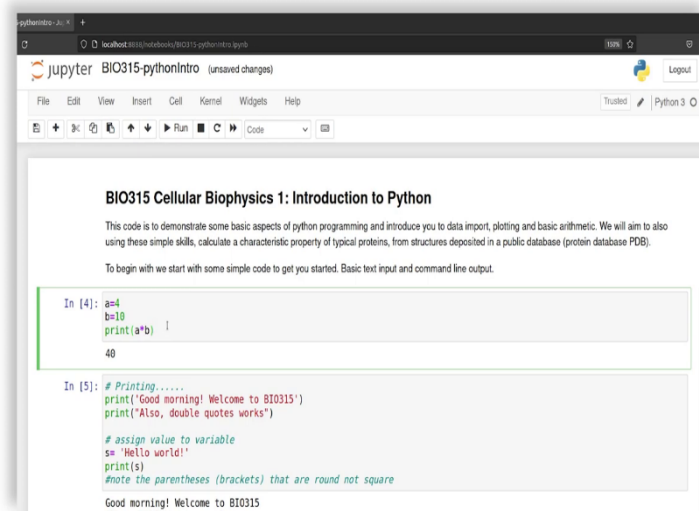
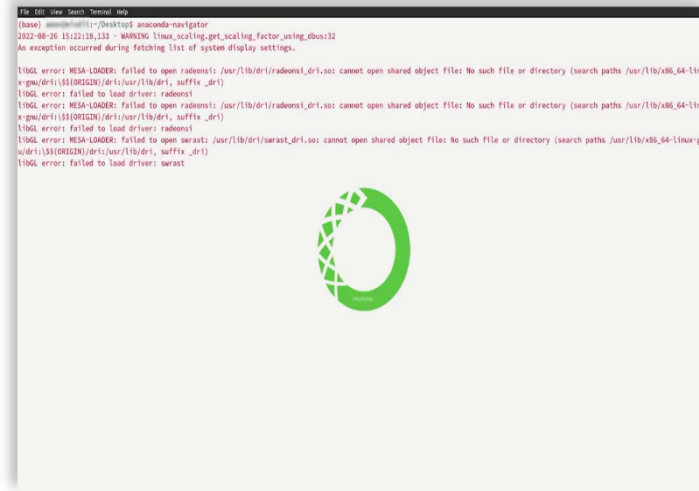
AIM: Basic concept, basic code, loops, I/O, graphing.

Assignment: Complete the plotting of histogram of the whole proteome dataset of E.coli. In class i will only demonstrate this for 9 proteins. You will need to modify my files for the purpose. These include protein abundance plots.

That is indeed an introduction to Python as a tutorial, to go over some basic concepts, writing some basic code, and being able to manipulate loops, file input output, and graphing. Your actual assignment is to complete the plotting of a histogram of the whole proteome data set of E.coli. I will demonstrate this for only 9 proteins, and you need to modify my files which I am going to provide to you to plot it for the whole proteome.

Usually, in a live class I actually go over each of your code and explain to you what the problem is, but once you submit the code, and if it runs, then I will give you feedback if there are any issues, and if it does not run, then obviously we will be discussing the answer in class. So, with all that in mind let us get back straight to it because coding is much more fun than talking about coding.

(Refer Slide Time: 02:05)



So, the way I run Python on my machine is through Jupyter notebook. In order to get this, I went to my command line interface where I typed anaconda navigator and that triggered a navigator window, and what it does is amazingly enough it opens a browser window, and this is what you see here.

This code again as I said is being shared with you, so that should hopefully not be an issue. What you need to notice here is that, I hope you see some nicely formatted text here, but if I click on it, it is called mark down. So this is a browser, Firefox is what I use. Markdown is nothing but text, highlighted text.

There is a syntax to, it a so-called second level heading is marked by double hash signs, and BIO315 Cellular Biophysics, it highlight, auto highlights the text, and now I have written some explanation about what I hope this exercise is going to show you, that is to demonstrate basic aspects of python programming, introduce you to plotting, and basic arithmetic, and then characterize the protein database entries.

So, we have to start somewhere, let us start with the simplest. Just aside, so far, these were markdowns, this is now code. So this you see over here in this window that the cell, in other words Jupyter divides the code flow into cells contains some elements, they can be either code, markdown, raw NB convert or headings.

And we are only going to be concerned with code and markdown, if you are interested, please go back and look what raw NB convert and headings means. As I said in my introduction, Python has extensive documentation and that is what makes learning as you go so much easier. I know some of you on your phones, so this might be a little bit harder but there is no rush.

So, in order for me to convert this marked out this raw ugly looking text into nice beautiful formatted text, all I need to do is press run. So now, it ran just this cell, so the code is divided into cells and those cells are independently run, they can also be connected. So, if I run 1 cell and then had some information in which I want the next cell to run, then obviously run them either together in sequence at least.

So, let us look at the first cell here. I set a is equal to b is equal to 4, b is equal to 10, and use a magnificent giant computer as a simple calculator, and of course I get the answer of a times b, this asterisk sign indicates multiplication, hence 40. All good. So now I move on.

(Refer Slide Time: 05:00)

The screenshot shows a Jupyter Notebook interface with the following content:

```

In [5]: # Printing.....
print('Good morning! Welcome to BIO315')
print("Also, double quotes works")

# assign value to variable
s = 'Hello world!'
print(s)
#note the parentheses (brackets) that are round not square

Good morning! Welcome to BIO315
Also, double quotes works
Hello world!

```

Why write a computer programme for biological problems?

1. **Simplify** Labour intensive , Time consuming , Error prone , Subjective More automated , universal , confidence. Data collection , analysis and representation easier
2. **Data analysis** Data comprehensible , observe trends (similarity or difference) Qualitative to quantitative measures Statistical analysis Model fitting , interpolate & extrapolate data Propose hypothesis, construct models and test
3. **In silico experiments** Modelling and simulation Numerical analysis Often cumbersome, expensive, difficult to do perform experiments hence can be predictive.

I am going to just sort of discuss a little bit of printing, what else can you print? And this is you know print hello world is the first line that Kernighan and Ritchie put into their textbook on C. And so Welcome to BIO315 is my message to you, hello world, just for fun, and suffice to say, these are single quotation marks, these are double quotation marks, both of them work.

(Refer Slide Time: 05:32)

The screenshot shows a Jupyter Notebook interface with the following content:

Why write a computer programme for biological problems?

1. **Simplify** Labour intensive , Time consuming , Error prone , Subjective More automated , universal , confidence. Data collection , analysis and representation easier
2. **Data analysis** Data comprehensible , observe trends (similarity or difference) Qualitative to quantitative measures Statistical analysis Model fitting , interpolate & extrapolate data Propose hypothesis, construct models and test
3. **In silico experiments** Modelling and simulation Numerical analysis Often cumbersome, expensive, difficult to do perform experiments hence can be predictive.

What can you do with your code

- automated plotting script
- script for regression analysis (curve fitting)
- perform statistical analysis
- model and simulate your data or hypothesis

```

In [ ]: # This is a full-fledged programming language. But you are free to
# use it like massive calculator, to link strings,
# and do simple input-output operations
## -----
# Add

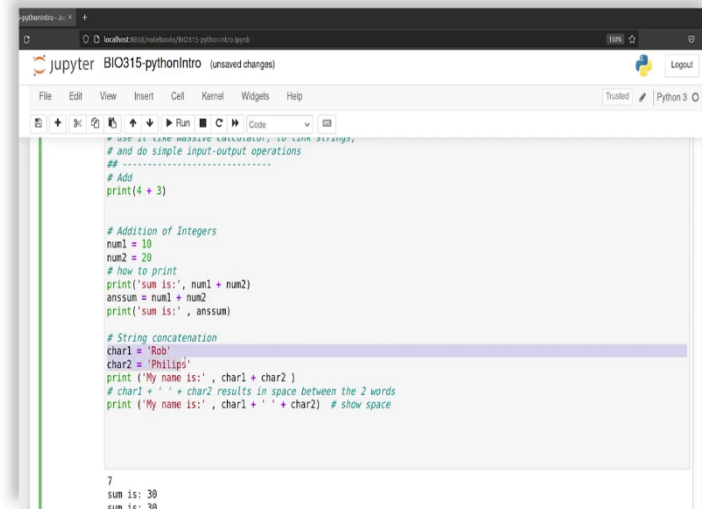
```

I talked about this in class, why do we bother writing a computer program for biological problems? The idea is of course clear. Simplify, what is a labour intensive, time consuming, error prone, subjective task, make it more automated universal, improve confidence, data collection, analysis, representation is easier.

And I talked about how you can do analysis, and in silico experiments. So, I am going to go ahead a little bit faster. What you can do with the code is actually automated plotting, and

scripting, script for regression analysis, perform statistical analysis, modelling, simulator data or hypothesis.

(Refer Slide Time: 06:03)



```
pythonIntro - Jupyter
localhost:8888/notebooks/BIO315-pythonIntro.py?

jupyter BIO315-pythonIntro (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

# Add: 4 + 3
print(4 + 3)

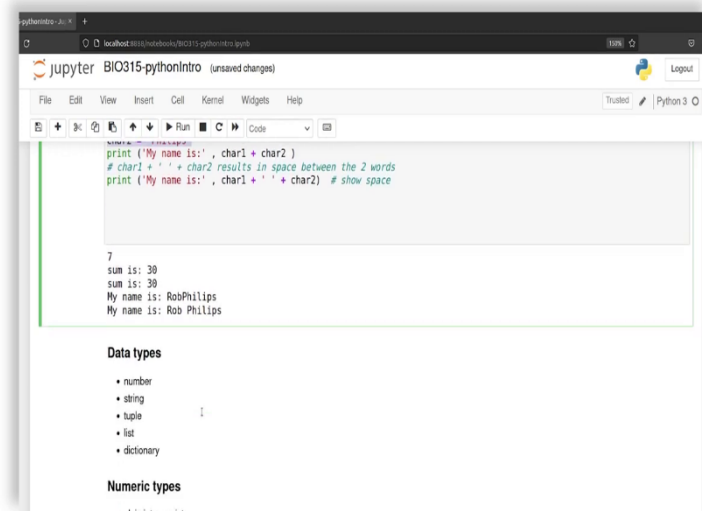
# Addition of Integers
num1 = 10
num2 = 20
# how to print
print('sum is:', num1 + num2)
anssum = num1 + num2
print('sum is:', anssum)

# String concatenation
char1 = 'Rob'
char2 = 'Philips'
print('My name is:', char1 + char2)
# char1 + ' ' + char2 results in space between the 2 words
print('My name is:', char1 + ' ' + char2) # show space

7
sum is: 30
sum is: 30
```

Indeed, while we can use it as a trivial calculator of adding numbers, and taking some outputs, we can do a bit more. So, as you see the sums over here are fine, and I can even do the same thing with strings, in other words, words or alphabets that are strung together to create words and even sentences, if I want.

(Refer Slide Time: 06:31)



```
pythonIntro - Jupyter
localhost:8888/notebooks/BIO315-pythonIntro.py?

jupyter BIO315-pythonIntro (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

# Add: 4 + 3
print(4 + 3)

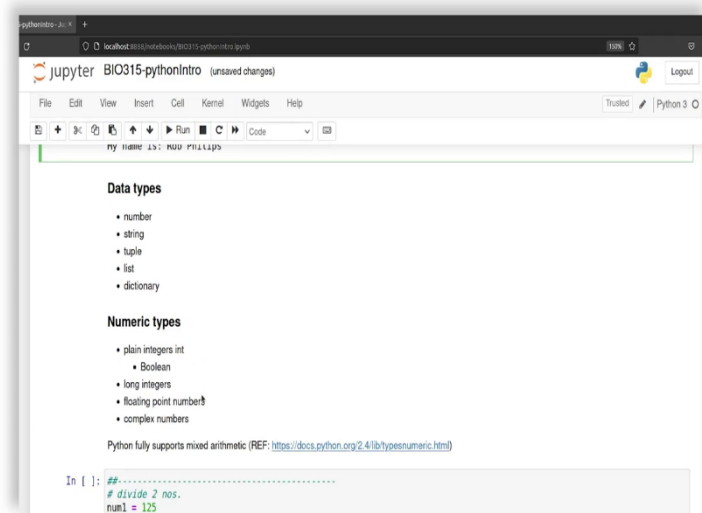
# Addition of Integers
num1 = 10
num2 = 20
# how to print
print('sum is:', num1 + num2)
anssum = num1 + num2
print('sum is:', anssum)

# String concatenation
char1 = 'Rob'
char2 = 'Philips'
print('My name is:', char1 + char2)
# char1 + ' ' + char2 results in space between the 2 words
print('My name is:', char1 + ' ' + char2) # show space

7
sum is: 30
sum is: 30
My name is: RobPhilips
My name is: Rob Philips

Data types
• number
• string
• tuple
• list
• dictionary

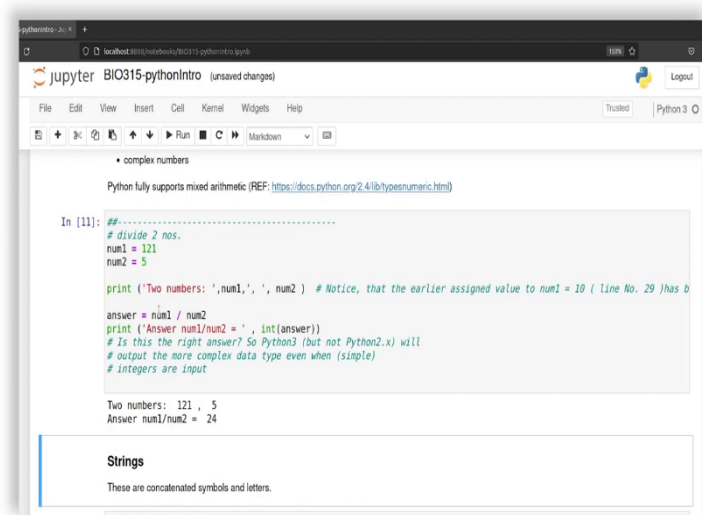
Numeric types
• add-in intname int
```



So, indeed this is important at this point to bring up the fact that the data types that we are interested in are number, string, tuple, list, and dictionary, and number types of numeric types are plain integers int, Boolean, long integers, float point numbers, complex numbers. Booleans that are some type of in zeros and ones.

I would really urge you if you want to learn more about the numerical support provided by Python to refer to the Python documents on numeric types. so, you can also of course do division, so addition, multiplication, subtraction, division.

(Refer Slide Time: 07:07)

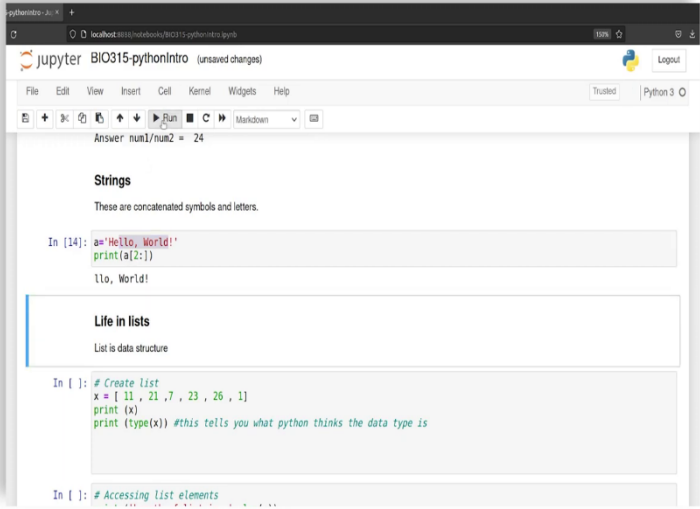


Now, the question is depending on the version of Python you are running and Google collab for coding, does have the latest Python 3, but if you had a local version of Python 2 point something, you will output a more complex data type even when simple integers are input,

meaning to say, that if I now instead of 12 by 5, put 13 by 5, and run it, yes, indeed I need to change it here.

So instead of 125, if I put 121 here, so what you are seeing here is a truncation to the integer output, so this answer is not quite right. And we need to somehow find a way to get hold of it, and I leave this to you, and we will discuss it on our discussion session. What must you do to get the full decimal numbers or the so-called float values beyond the integer, if you give a so-called not fully divisible number? Strings on the other end, are concatenated symbols and this is an example over here of hello world.

(Refer Slide Time: 09:19)



The screenshot shows a Jupyter Notebook window titled "jupyter BIO315-pythonIntro (unsaved changes)". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for running, saving, and other actions. The notebook content is as follows:

```
Answer: num1/num2 = 24
```

Strings
These are concatenated symbols and letters.

```
In [14]: a = 'Hello, World!'
print(a[2:])
llo, World!
```

List in lists
List is data structure

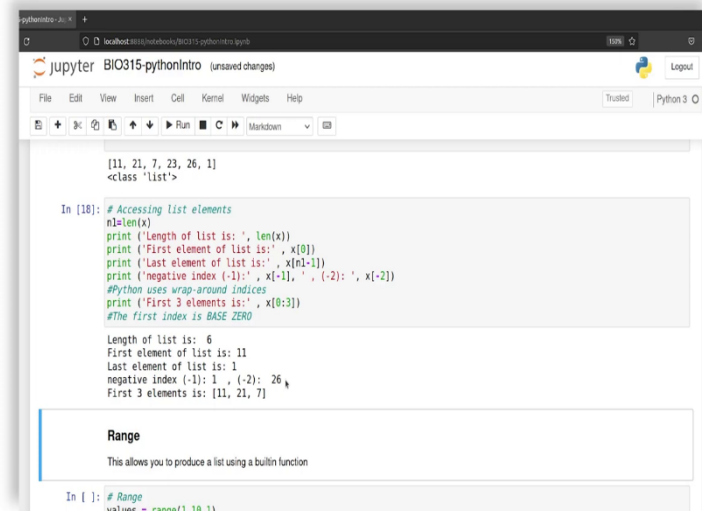
```
In [ ]: # Create list
x = [ 11 , 21 , 7 , 23 , 26 , 1]
print (x)
print (type(x)) #this tells you what python thinks the data type is
```

```
In [ ]: # Accessing list elements
```

Indeed, strings have also positions, and as I said in my overview, it is always 0, 1, 2, 3, 4, 5, 6, and then 7, 8, 9, 10, 11, 12. So, if I say a from the 7th element, that is base 0 onwards to the last element signified by this semicolon sign, will give you just the second part world. If I now modify this to say from 2 onwards for instance, you should see in fact 0, 1, 2 onwards, low, the lower. So, you can play around with this. Please use it.

Indeed lists, which is kind of what we want to use for initially at least storing some of our data is a nice simple structure that kind of looks like an array but it is not an array, because it is one-dimensional whereas arrays can be n-dimensional, and as this variable states, if I put in some numbers 11, 21, 7, 23, 26, 1, 0, 1, 2, 3, 4, 5, that is 6 in length, and if I print it, I will actually get the numbers again. And type is the data type it is, and this actually tells me from the code that it is actually list.

(Refer Slide Time: 10:23)



```
[11, 21, 7, 23, 26, 1]
<class 'list'>

In [18]: # Accessing list elements
n=len(x)
print ('Length of list is: ', len(x))
print ('First element of list is: ', x[0])
print ('Last element of list is: ', x[n-1])
print ('negative index (-1): ', x[-1], ' (-2): ', x[-2])
#Python uses wrap-around indices
print ('First 3 elements is: ', x[0:3])
#The first index is BASE ZERO

Length of list is: 6
First element of list is: 11
Last element of list is: 1
negative index (-1): 1 , (-2): 26
First 3 elements is: [11, 21, 7]

Range
This allows you to produce a list using a builtin function

In [ ]: # Range
values = range(1, 10, 1)
```

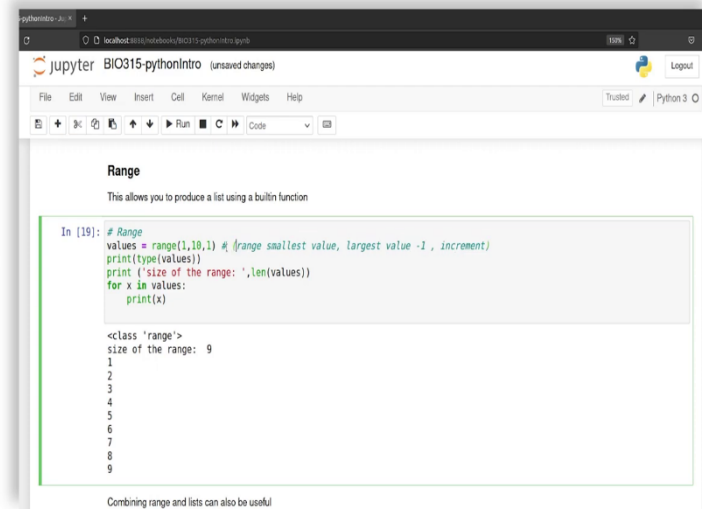
List elements can of course be individually accessed. I can call the length of the list, this is the command `length list` or `length of that list` in brackets, I can invoke certain elements of that value, the zeroth value for instance, the first in that sense. I can even call other values of it, so let us say I want to find out the last value. Will this work?

So let us see. My last value remember is 1. So, if the length is 7, the last value throws an error. This, remember, is because of our base 0 rule. Now, suddenly my last value is correct. You can also use the so-called negative index, which is, so what do you think the negative index will give you?

So if you look here of the value at `x` of minus 1 is 1, in other words it is the last value, minus 2 is the last but 1 value which is 26, and so on and so forth. Try it out yourself and you will find out. In that sense, the indices are wrapped around. You go around and you go minus 1, minus 2, minus 3, etc, etc.

That is not base 0, yes, you are right. The first three elements for instance or the first `n` elements are 0 to something, that is why the colon, the semicolon. And I hope you remember this that that we had used the colon earlier to give an open-ended range, also everything from something to something else.

(Refer Slide Time: 12:55)



```
Range
This allows you to produce a list using a builtin function

In [19]: # Range
         values = range(1,10,1) # (range smallest value, largest value -1, increment)
         print(type(values))
         print('size of the range: ',len(values))
         for x in values:
             print(x)

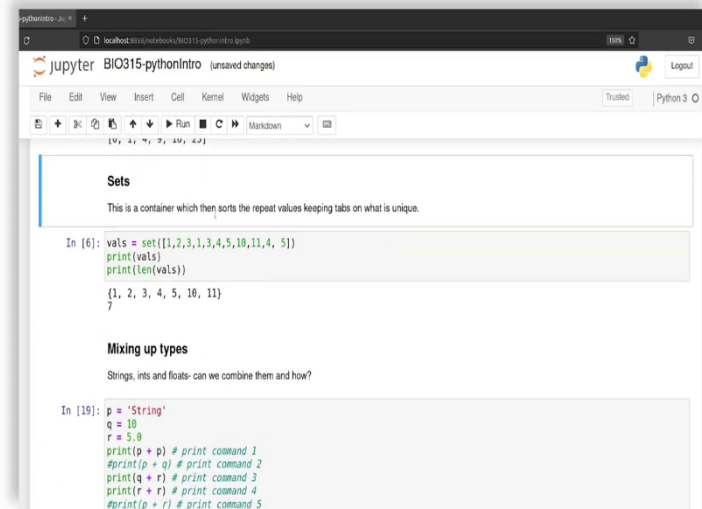
<class 'range'>
size of the range: 9
1
2
3
4
5
6
7
8
9

Combining range and lists can also be useful
```

Range is a built-in function which allows you to create a series of numbers in a, let us find out what kind of structure. It is a range and the size of the range is indeed its length. And for each value in that range, we can actually iterate, in other words, we can pass values to the for loop, it will just simply go through every element from 1 to 10 with a step of 1 to list it out. Now, you notice that 10 is a non-inclusive limit, in other words, it stops 1 minus that or 1 increment above.

So you can say here we have range, smallest value, largest value, minus increment, and increment. Meaning to say, increase by 3, increase by 2, increase by n, that value minus the largest value is what the last value of the, I am sorry, plus the increment will be as it is output, okay. So, in other words if I want to range from 1 to 10, actually including 10, then I need to add 1 here, and then I will get my 1 to 10.

(Refer Slide Time: 14:47)



The screenshot shows a Jupyter Notebook interface with two code cells. The first cell is titled 'Sets' and contains a Python code snippet that creates a set from a list of numbers, including duplicates, and prints the set and its length. The output shows the set {1, 2, 3, 4, 5, 10, 11} and its length 7. The second cell is titled 'Mixing up types' and contains a Python code snippet that defines variables p, q, and r with different data types and prints their concatenations. The output shows the concatenation of 'String' and 10 as 'String10', 'String' and 3.0 as 'String3.0', and the concatenation of 10 and 3.0 as '103.0'.

```
In [6]: vals = set([1,2,3,1,3,4,5,10,11,4, 5])
print(vals)
print(len(vals))
{1, 2, 3, 4, 5, 10, 11}
7
```

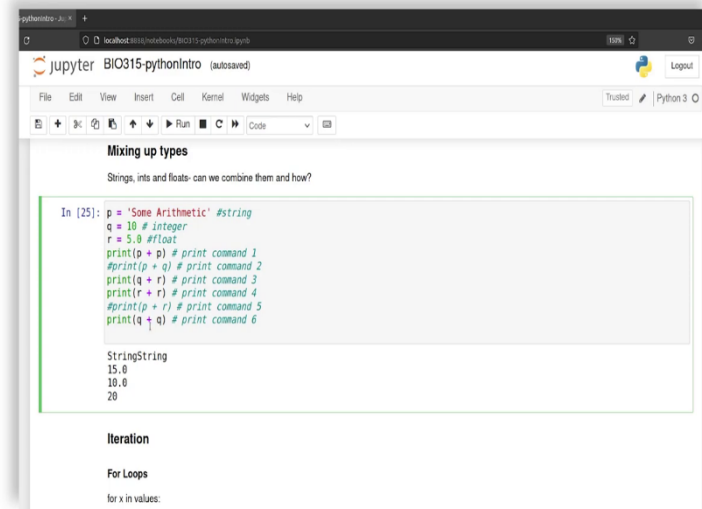
```
In [19]: p = "String"
q = 10
r = 3.0
print(p + p) # print command 1
#print(p + q) # print command 2
print(q + r) # print command 3
print(r + r) # print command 4
#print(p + r) # print command 5
```

So, combining ranges and lists cannot be useful, so I can say x double star 2 for x in range 6, so x in range 6 is like we did here. So let us look at this. Range 6 is 0 to 6 with increments of 1. x double star asterisks 2 is square of that. So 0 is 0, 1 is 1, 2 is 4, 3 is 9, 4 is 16, 5 is 25, familiar to you. So it is a very quick way to get us a range containing some arithmetically manipulated values.

Sets are a container which then sort and repeat the values keeping tabs are not as unique. Now here, you see a set which is 1, 2, 3, 1, 3, 4, 5, 10, 11, 4, 5. What you should hopefully notice is there are a few redundant numbers meaning to say repeated numbers. So, when I just print the values, what do you think I get?

So, if I, if it was an ordinary list, then I would get everything, yes, which is this. But set finds the unique values in that, and puts it out. In other words, the length of the set 1, 2, 3, 4, 5, 6, 7 is 7, but the length of the list is 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11. So I can even maybe create a list and attribute it to set and try and find the so-called unique values.

(Refer Slide Time: 16:45)



The screenshot shows a Jupyter Notebook interface with the following content:

```
Mixing up types  
Strings, ints and floats- can we combine them and how?  
  
In [25]: p = 'Some Arithmetic' #string  
q = 10 # integer  
r = 5.0 #float  
print(p + p) # print command 1  
#print(p + q) # print command 2  
print(q + r) # print command 3  
print(r + r) # print command 4  
#print(p + r) # print command 5  
print(q + q) # print command 6  
  
StringString  
15.0  
10.0  
20
```

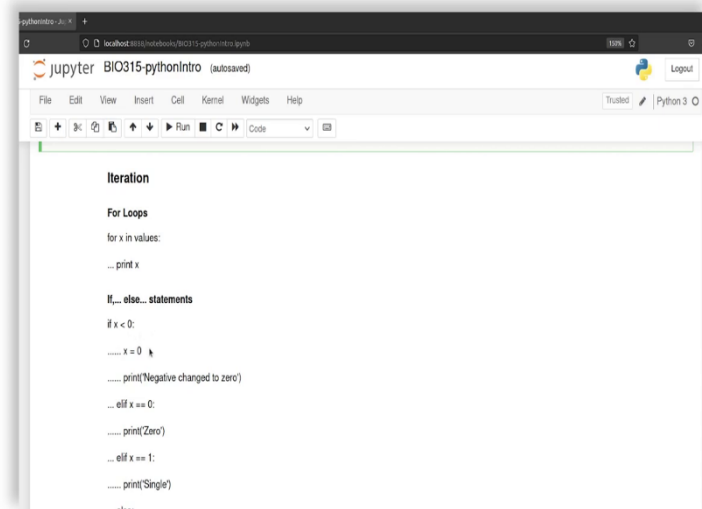
Below the code cell, the notebook displays the following text:

```
Iteration  
  
For Loops  
for x in values:
```

You can also mix subtypes, strings, integers, and floats, and combine them. So, for instance, I want to combine the word string with an integer and a float, and let us see if it works. So, string gives string, string, string, b plus p, p plus q, which is string plus integer gives me what? Let us see. An error. So that does not exactly work.

q plus r gives me a command, which does indeed work, q plus r, so integer, and string can be added, I am sorry, integer and float can be added, but a string cannot be added to either an integer or a float, as we saw earlier. Try it out yourselves if you do not want to believe me. And this is the beauty of it, I want you to be able to try these things out.

(Refer Slide Time: 18:13)



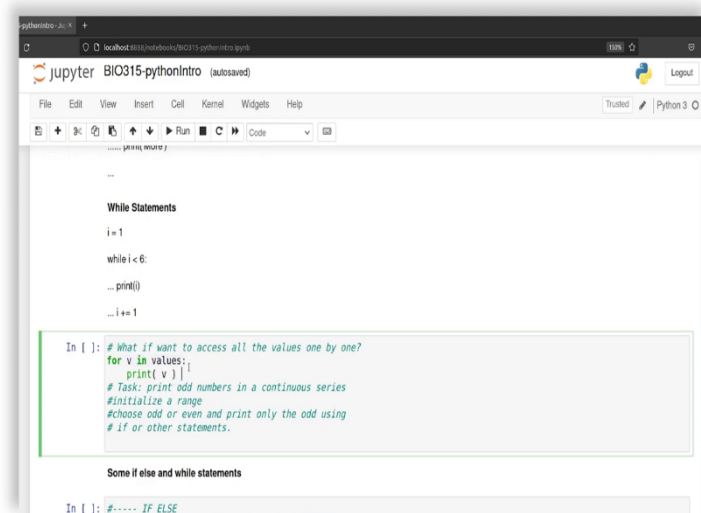
The screenshot shows a Jupyter Notebook interface with the following content:

```
Iteration  
  
For Loops  
for x in values:  
... print x  
  
If... else... statements  
if x < 0:  
... x = 0  
... print("Negative changed to zero")  
... elif x == 0:  
... print("Zero")  
... elif x == 1:  
... print("Single")  
... else:
```

Now, what you see in the last bit, I want to talk about over here, are for loops, and if else loops and while statements. In other words, something that iterates. So, for statements are given as x in values something. So, in some either a range or a list or something. This will allow you to print that value. If else statements need to be structured as some criterion, some good comparison.

If x is less than 0, do something, elif not else, if but l, if x is greater equal to something, colon, print 0, lf, and then so on and so forth, or while statements which are a little more dangerous, unless you use them carefully, where you initialize your value i is equal to 1, while i is less than something, print I, and incremented so that it stops, just 1 shot of that. So, this is also, these are the standard loop, looping syntaxes which you have seen probably in various places.

(Refer Slide Time: 19:18)



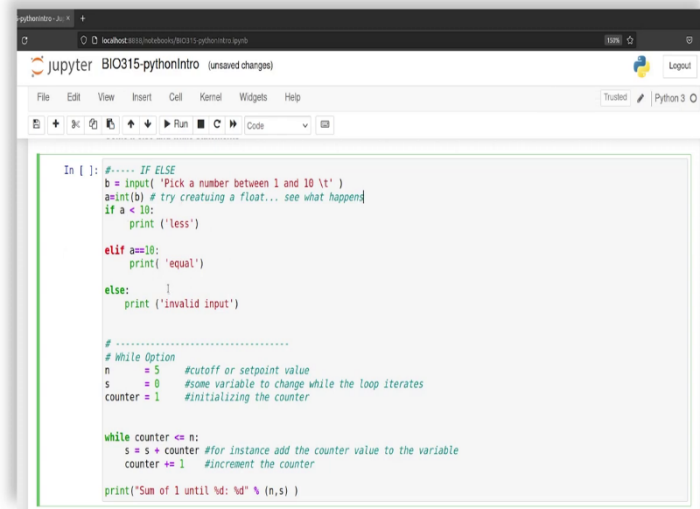
```
..... print(winner)
...
...
While Statements
i = 1
while i < 6:
... print(i)
... i += 1

In [ ]: # What if want to access all the values one by one?
for v in values:
    print(v)
# Task: print odd numbers in a continuous series
# initialize a range
# choose odd or even and print only the odd using
# if or other statements.

Some if else and while statements
In [ ]: #..... IF ELSE
```

What if we want to access some values 1 by 1 that are in an array? Remember, we created a list, I am sorry, a list somewhere called values. So, we said values are a range from so and so to, so and so. So let us see if we can access each of those values using our for loop. And low and behold by just simply saying v for v in values which is that list that we gave it from range, using the function range, we could actually access each 1 by just simply invoking an element in that. Iteratively, the iterator for 'For' actually is very clever in that sense in Python.

(Refer Slide Time: 20:05)



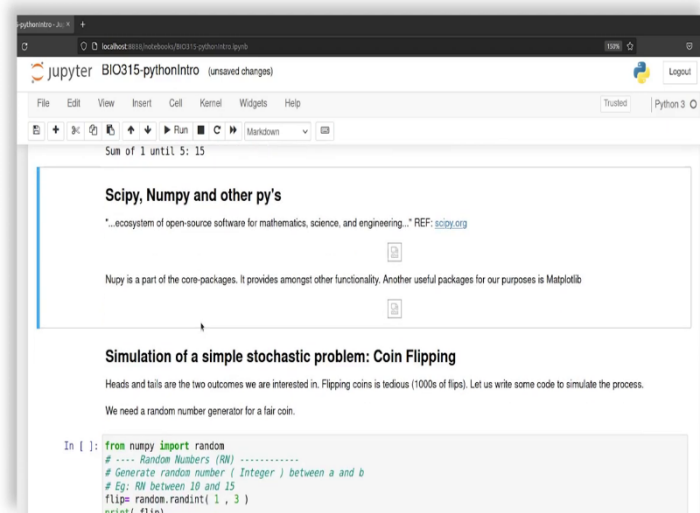
```
In [ ]: #..... IF ELSE
b = input( 'Pick a number between 1 and 10 \t' )
a=int(b) # try creating a float... see what happen
if a < 10:
    print( 'less' )
elif a==10:
    print( 'equal' )
else:
    print( 'invalid input' )

# -----
# While Option
n = 5 #cutoff or setpoint value
s = 0 #some variable to change while the loop iterates
counter = 1 #initializing the counter

while counter <= n:
    s = s + counter #for instance add the counter value to the variable
    counter += 1 #increment the counter
print( 'Sum of 1 until %d: %d' % (n,s) )
```

So, now if something else and while statements can be easily structured. You think of it logically, pick a number between something, create some integer value of that number, meaning to say, make it into a type that we can deal numerically, you can either create it as integer, try creating a float out of it, and see what happens. And, if that value that you input you yourself or your friend who you pass this code on to, is less than some criterion, print less, greater than some criterion, equal, or you want to say invalid input, that is what you do.

(Refer Slide Time: 21:02)



```
Sum of 1 until 5: 15
```

Scipy, Numpy and other py's

"...ecosystem of open-source software for mathematics, science, and engineering." REF: scipy.org

Numpy is a part of the core-packages. It provides amongst other functionality. Another useful packages for our purposes is Matplotlib

Simulation of a simple stochastic problem: Coin Flipping

Heads and tails are the two outcomes we are interested in. Flipping coins is tedious (1000s of flips). Let us write some code to simulate the process.

We need a random number generator for a fair coin.

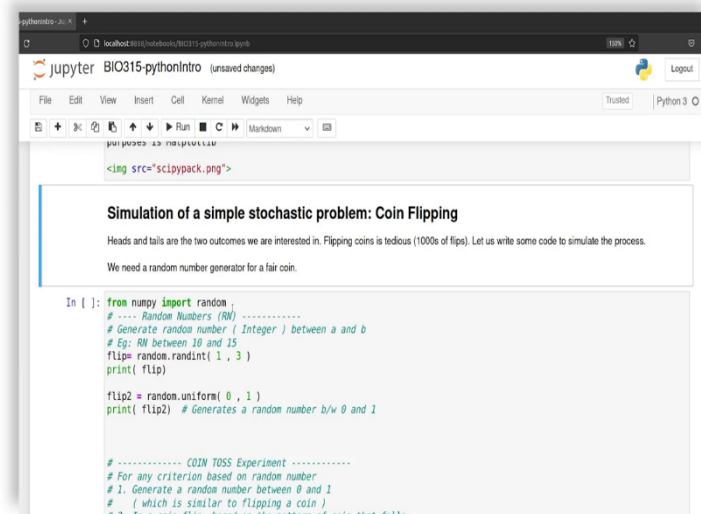
```
In [ ]: from numpy import random
# ---- Random Numbers (RN) -----
# Generate random number ( Integer ) between a and b
# Eg: RN between 10 and 15
flips= random.randint( 1, 3 )
print( flip)
```

So, let us see if this runs, pick a number between 1 and 10, it has to be integer according to my definition, so I picked 4, it is indeed less than 10, sum of 1 until that number is what I calculated using my while. So, while the counter is less than or equal to n, where n is some

cut off a set point value, then just count the number of increments. So how many values until I reach my criteria.

There are many Python packages, so scipy, numpy, and other pys, and you can read more about them on scipy.org, for scipy, and numpy is a part of the core packages, has numerous functionalities, numpy, scipy, library, matplotlib, consoles, and data structure, and data import output, packages like pandas.

(Refer Slide Time: 22:08)



```
python3 -> |>
jupyter BIO315-pythonIntro (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3


Simulation of a simple stochastic problem: Coin Flipping
Heads and tails are the two outcomes we are interested in. Flipping coins is tedious (1000s of flips). Let us write some code to simulate the process.
We need a random number generator for a fair coin.

In [ ]: from numpy import random
# ---- Random Numbers (RN) ----
# Generate random number ( Integer ) between a and b
# Eg: RN between 10 and 15
flip = random.randint( 1 , 3 )
print( flip )

flip2 = random.uniform( 0 , 1 )
print( flip2 ) # Generates a random number b/w 0 and 1

# ----- COIN TOSS Experiment -----
# For any criterion based on random number
# 1. Generate a random number between 0 and 1
# ( which is similar to flipping a coin )
# 2. For a coin flip, based on the random number, we can flip
```

So, we are going to start with the most simple exercise of a stochastic problem. And you remember I said we can do two things, we are going to basically analyze data, and we are going to simulate processes. For the analysis part, I would like you to focus on a stochastic problem of coin flipping, and I will talk about it next.