1. **Introduction**

1.1

Even if you have never programmed or you know the language go ahead with this course.

Python used for Web Development, Data Science, Machine Learning,

Automation, Art Projects.

Curriculum:

Installing

Command Line

IDE

Basics

Numbers

Variables and Data Types

Boolean

Rock Paper Scissors

Logic and Loop Related Games

Lists

Dictionary

Tuples

Sets

Functions Pt. 1 and 2 also exercises.

Lambdas

Debugging

Modules

External Libraries

Sec 25: Optional for Web Dev

Generators and Decorators

OOP

Deck of Cards Exercise

OOP Pt. 2

Test Driven Dev.

File IO

Iterators

Web Scraping And Beautiful Soup

(Crawls and extracts data from sites with our own spider)

Each Section has atleast 5 and maximum 20 exercises.

002 Python3 vs. Python2

In 2008 the developer of Python Guido van Rossum was unhappy with some architectural decisions made in Python 2 and he changed it.

Python 2.x is legacy Python 3.x is present and the future of the language.

Usually when there are changes made in a programming language then changes are made incrementally such that old code can still run.

But Python 3 isn’t backwards compatible.

We are 10 years into the future now and this mess between Py 2 and Py 3 is starting to get better but far from completely solved.

A big happy family Python 2.x everyone sending Hii and hearts to each other and then came Make Python Great Again motto in 2008 and Community split into 2.x and 3.x.

Py 2 was the best choice early on. But now pffft not even an option. Python 3 for the victory.

**Why Learn Py 3 ?**

Easiest to learn Py 3 and go back to Py 2 if and when needed.

Earlier most of the libraries and tools were written for Python 2 and if you went ahead with python 3 then you couldn’t get benefit of those tools.

But things have changed now and everything or even which is not available its better alternatives are available have found their way to the Python 3 community.

Out of 360 important packages 348 are supporting Py 3 now.

IT’s the FUTURE!!

In 2020 Py2 dies.

**003 How The Exercises Work**

Mostly exercises are in the last 75% of the course after Variables and Strings section.

**004 Where Should You Start**

If not comfortable with Powershell/ CLI then try CLI else normal Windows/ Mac installation.

**005 How The Course is Structured**

Some stuff is optional in a section starting exercises are easy and they get harder towards the end of the course.

**017 Why Do You Need The Command Line**

Get-Service | Where-Object {$\_.Status –eq “Running”}

python

help()

**018 Using Powershell**

Mac and Linux are same as Unix.

Windows has different kernel and different command lines and cmd prompts.

PowerShell use very similar not exact commands as Unix kernel.

**019 Paths LS and PWD**

Navigate around

Create and remove directories

Execute Python scripts

We use Powershell for the stuff written above.

OS File Structure

Root W: C: D: or other drives.

On PCs home directory is ~. C:\Users\Anmol

So in PowShell cd ~ takes you to C:\Users\Anmol

pwd print working directory.

ls shows list of where we are.

**020 Using The CD Command**

cd for change directory.

cd .. for go back one folder.

single dot .\Desktop means in current folder . go to Desktop folder

**021 OPTIONAL Making Directories with MKDIR**

mkdir folderName makes folder .

**022 OPTIONAL Making Files (its...complicated)**

In linux if we type touch.py in a directory then it makes the file touch.py.

In PowShell echo $null >> rps.py makes new file.

But python needs UTF 8 files and echo $null >> rps.py makes rps in UTF 16 which is not good.

**024 OPTIONAL Removing Files and Folders with RM**

rm filename.ext to remove any file in PowShell.

To delete a folder we use rm –r –fo directoryName which means remove the directory r stands for recursion and fo to force delete so that all warnings are ignored.

**030 Installing Python On a PC**

On typing python in PowShell we get Python version and other info which is called Python Interpreter called REPL. Read Evaluate Print Loop.

**031 Introducing Sublime Text**

Whole bunch of promotion about Sublime but I am going to go with VSCode.

**032 First Python Program**

Type python .\filename.py to run a file.

Python 2.x print “Hello!”

In Python 3.x it is a function print(“Hello!”)

“Hello” a string.

**034 Important Note on Print and the Python Interpreter**

Typing 5 and print(5) are two different things in the python interpreter.

Py Int is a place to quickly debug code to test things but no one writes code or long apps in the interpreter.

Py Inter takes one line and prints the output and again does the same.

**06 Numbers Operators and Comments Section!!**

**035 Section Introduction sObjectives**

Difference between Int and floats.

Math operators + - \* / % // etc.

Add Comments.

**036 Numbers Int and Floats**

Integer -23, 4 , 89, etc. have whole.

Float 6.1, 3.553234, 0.0 etc. have decimal.

Complex and long are other types of numbers as well.

python loads py inter. Then type(9) gives <class“int”>

type(9.0) gives <class“float”>

float + int = float.

20 + 0.77 = 20.77

**037 Numbers Basic Math**

+

-

\*

/

\*\* Exponentiation

% Modulo

// Integer Division

In Python ½ gives you 0.5 but in some programming languages like C, we would get 0 as it will give us only quotient value and nothing else.

In Python Division always returns float.

There is a hierarchy how these operations are evaluated in Python or any programming language.

1 + 2 \* 5 / 3

By BODMAS

PEMDAS Parentheses, Exponential, Multiply, Division, Addition, Subtraction

Therefore first multiplication. i.e. 10 /3 + 1

But (1+2)\*5/3 gives us 3\*5 then divided by 3. Giving us 5.0

**038 Comments**

Little notes inside our code to improve readability of our code.

Use # in python to comment.

**039 Numbers Weirder Operators**

Number \*\* To the power we have to raise. Exponential operator.

2 \*\* 3 gives 8. i.e. 2 to the power \*\* 3.

49 \*\* 0.5 gives 7.

Modulo % takes a number and divides other number to it as many times as it can before becoming fractional. % gives us remainder. 10 % 3 is 1.

25 % 6 is 1.

Useful to find if a number is even or odd.

// Integer Divison

Since python by default division of 2 ints gives float. To stop it from doing so we can use //.

10/3 = 3.33333

10 // 3 = 3 only returns int part of the quotient and eliminates the float part.

Integer division floors means it always rounds down.

**040 Python Documentation**

Way to go if you are ever stuck somewhere.

**07 Variables and Strings SECTION!!**

**041 Variables and Data Types**

Quick video about data types and kinds of data types but then move on to focus on strings.

Objectives :

How to assign and use variables.

Python variable naming and restrictions.

Different data types overview.

Why python is dynamically typed language

Understand how to convert data types.

Learn a lot about strings.

A silly program that gets user input.

**042 Intro to Variables**

Variables are like containers

Store some data pull it out later

They can hold all sort of things

Like numbers, Booleans and strings.

Named symbol that holds a value. Just like algebra. A jar with a label holding whatever.

In python variable initialization looks like following

X = 1000

Or khaleesi\_mother\_of\_dragons = 1

We aren’t just limited to storing numbers in variables but all sorts of data.

We can recall those variables using

print(khaleesi\_mother\_of\_dragons + x)

# Gives output as 1001.

Variables make program legible and reusable.

Most of our data is not static and depends on user input.

Code should work with dynamic data.

Variable Assignment

Variables can be assigned to other variables as well at the same time.

python\_is\_awesome = 100

just\_another\_var = python\_is\_awesome

Multiple variable assignment

all, at, once = 5, 10, 15

assigns following values all = 5 ; at = 10; once = 15

**044 Naming Restrictions**

Info on what we can and what we can’t name our variables in Python.

Also stylistic conventions of what you should name variables.

Rules:

1. Variables should start with an alphabet or underscore. Not with a bracket, number or question mark.
2. Rest of the name must consist of letters, numbers or underscores.
3. Names are case sensitive.

CATS != cats != Cats in python. They all are different.

Naming Conventions

Won’t break your program but a good practice to follow what everyone else is doing.

Variables in python should be snake\_case that is using underscore between words and keeping everything small but in python land we do not use camelCase. Python is a snake and thus uses snake\_case.

Most variables should be lower case. But when assigning value to something which is constant always throughout the program then we use CAPITAL\_SNAKE\_CASE.

UpperCamelCase refers to a class while writing Python code.

Variables starting and ending with 2 underscores are called dunder or double underscores and refer to stuff which has to be left private or \_no\_touchy\_

Dunder are found when working on open source projects and the author of the code just means that please these dunders are not for you to play/mess with.

You can change it. It’s like your mom telling do not eat all the cake, but there’s no one stopping you to do so and face the consequences.

**045 DATA TYPES**

Numbers to store price, stock and other things.

Name or date or description can also be stored in variable.

And we use other Data Types to store other things.

Most common ones are

|  |  |
| --- | --- |
| **Data Type** | **Description** |
| Bool | True or False |
| Int | An Integer (-3, -2, 0, 1, 2, 3) |
| Str | String a Sequence of Unicode Char. E.g. “Cosmic Commander” “adsad55” |
| List | An ordered sequence of values of other data types [1, 2, 3] [“a”, “b”, “c”] |
| Dict | A collection of key values {“first\_name”: “Colt”, “last\_name”: “Steele” |

Bool : Yes or No. Takes much less space as compared to int because 0 or 1.

Python understands True and not true or TRUE.

is\_active = True

on entering is\_active returns true in the console.

Same for False and not false or FALSE.

Python recognizes on the basis of capital T or F of True and False respectively and we can’t do t or f or all CAPS as well.

Strings are just characters and in quotes single or double.

Dictionaries have pair of corresponding values stored together.

Dict and List are data structures that store other types of data.

**046 What the heck is Dynamic Typing**

Python is highly flexible about reassigning variables to different types.

Gives us ability to reassign our variables, not just from changing values within one single data type but also allows us to go from 99 to 98 in number data type to 99 from number to False of Boolean or None or Zero of String data type in one single variable itself.

Python allows variables to change type.

awesomeness = True Boolean

print(awesomeness) #True

awesomeness = “a dog” String

print(awesomeness) #a dog

awesomeness = None None

print(awesomeness)

awesomeness = 22/7 Float

print(awesomeness) #3.142857…

None is the concept that python represents to signify literally nothing.

Languages like C++ are statically typed i.e. once you define a variable of a particular data type then you can’t change it’s data type by assigning something else to it later on.

We can’t really dynamically change in statically typed language.

**047 The Special value None**

If you type None in Py Inter you get nothing. That’s what it is. It’s nothing.

Just like True or False, None makes sense to python and none or NONE means error.

It is helpful in the case when you have a variable like a form and if someone leaves a field then instead of assigning NO we can just allow users to have that field unanswered and set it to None in the cases where no answer was reported.

name = “Daisy”

age = 30

child = None

on typing print(child) we get None. It’s a better option instead of writing empty string ‘’ or zero or false we keep it None.

None signifies there is a container for if and when Daisy haves child but for now that child container has nothingness represented by None.

type(child) gives NoneType.

**048 Double Vs. Single Quotes**

We’ll talk about strings for now and will visit dict , list later on as well.

Different ways to declare strings:

We can go with single or double quotes but it does not matter at the end of the day.

But you have to be consistent.

You can use ‘’ or “” but you cannot use ‘” or “’.

Stick to the same convention throughout the file.

It’s a style thing like snake\_case in python.

When “” or ‘’ any of the string is returned back they are returned in ‘’ single quotes and look exactly the same.

We can use quotes inside of other quotes.

msg = “He said “This is really good” ”

Now when we print(msg) then python reads till first 2 “ and then prints He said and quits throwing an error after that because it gets confusing.

What we have to do is that we can use different type of quotes

msg = “I am ‘funny’ ” or ‘I am “funny” ’

**050 String Escape Sequences**

In python there are also “Special Characters” which are “metacharacters” – They get interpreted by Python to do something special.

Classic example is that of a return inside a string.

Most of the escape sequences start with backslash.

\\ Backslash.

\’ Single Quote

\” Double Quote

\b ASCII Backspace

\n New Line

We can include hexadecimal characters with \x

\t ASCII Horizontal TAB.

str = “This is a backslash \” Error!

str = “This is a backslash \\”

print(str)

This is a backslash \

**052 String Concatenation**

Concatenation is combination of multiple strings together.

In python we can do this simply with the + operator.

str\_one = “your”

str\_two = “face”

str\_three = str\_one + “ “ + str\_two

print(str\_three) gives us

your face.

username = "Blue\_The\_Cat"

>>> print("Hello there and welcome to the game, " + username)

Hello there and welcome to the game, Blue\_The\_Cat

>>> str\_one = "ice"

>>> str\_one += " cream"

>>> print(str\_one)

ice cream

+= or -= \*=isn’t limited to just strings infact applicable to numbers as well.

>>> people = 99

>>> people += 1

>>> people

100

**054 Formatting Strings**

We have data and some strings and we put them together goes to the domain of concatenation.

Either to display a message or create a new variable do something with that combination.

There are other options instead of just concatenating single or individual strings.

What if we are working with numbers, like I wanted to be able to say your guess and then add something to it. Solution to this problem is interpolation.

To interpolate variables it means that

Formatted strings

The new way in (Python 3.6+) => F-strings

x = 10

formatted = f“I’ve told you {x} times already!”

Taking the variable x and interpolating inside curly braces.

>>> guess = 8

>>> print(f"Your guess of {guess} was incorrect!")

Your guess of 8 was incorrect!

Can also do math.

>>> guess = 8

>>> print(f"Your guess of {guess + 1} was incorrect!")

Your guess of 9 was incorrect!

>>> name = "bluethecat"

>>> guess = 8

>>> print(f"Nice try {name} but your guess of {guess} was incorrect!")

Nice try bluethecat but your guess of 8 was incorrect!

**057 Strings and Indexes**

Each string is zero indexed. i.e. each character is assigned a number.

“lol” 0 = l, 1 = o, 2= l

We access these specific characters by [] putting the number of those characters inside a string.

“China”[2] gives ‘i’ as output.

>>> name = "Norris"

>>> name[5]

's'

These [] are part of index lists and indexed items are items being stored inside them.

Useful in an app where at the end we say users if they want to play or not and we ask them to reply in yes, yaa, yeah, yup, or no, noii, nien, nope etc. then we would only inspect [0] char. To see if it’s y or n and proceed as per the choice.

[-ve] Means that it starts from the last. -1 being the most last letter of string.

>>> answer= "YesMyFriend"

>>> answer[-2]

'n'

>>> answer[-3]

'e'

>>> answer[-9]

's'

Indices in python always start at zero 0.

**058 Converting Data Types**

How to take an integer and turn it into a float and vice versa ?

Or maybe we have a string of number 8 inside double quotes and we want to change those into number 8 or vice versa ?

It’s pretty straightforward.

When we had interpolation data types are implicitly converted into string form.

We can also convert variables explic qitly by using the name of the built in type as a function.

>>> decimal = 12.5646546

>>> integer = int(decimal)

>>> integer

12

>>> my\_list = [1, 2, 3]

>>> my\_list\_as\_a\_string = str(my\_list)

>>> my\_list\_as\_a\_string

'[1, 2, 3]'

>>> num = 12

>>> type(num)

<class 'int'>

>>> float\_num = float(num)

>>> float\_num

12.0

>>> str(8)

'8'

Don’t make variable names as str, int, float, or print.

**059 Building a Mileage Converter**

How to get user input in Python ?

We use input() just a function like any other function such as print() or type() that takes its input inside parentheses, int(), str() etc.

Doesn’t asks anything just waits for user to type something and hit the return/Enter key.

We can name the input in the following manner If var name if kms

kms = input()

print("How many kilometers did you run today ?")

kms = *float*(input())

miles = kms/1.60934

miles = round(miles, 2)

print(f"Your {kms} km ride is equal to {miles} miles." )

#round(thing to round, places to round to)

#we can do {round(miles, 2)} . Inside {} curly braces we can do math or whatever logic we want.

Round(thing to round, how many decimal points to round it to)

Round is a function.

**08 Boolean and Conditional Logic**

**060 Section Introduction and Objectives**

True or False values. About making decisions and really really important.

Python ain’t no flawed human. It is Yes or No. 0 or 1.

True or False. No in between.

**Objectives:**

Learn how to get user input.

Learn about truthiness. Values are truthy or falsy.

Learn how to use comparison operators to make a basic program.

**061 Getting User Input**

There is a built in function in Python called input() that prompts the user to enter a value.

If user types something and press enter it is stored in a var.

We can at the same type print and pass input by doing the following:

name = input(“Enter your name: ”)

# Enter your name:

>>> name = input("Enter your name here: ")

Enter your name here: Cosmic

>>> name

'Cosmic'

**062 Boolean Expressions**

If : The way of implementing the different forking paths in our programs based off of comparisons.

Conditional logic using if statements represents different paths a program can take based on some type of comparison of input.

Python is very whitespace sensitive that’s how it knows which part belongs to where.

Which part belongs to if, which belongs to else and which is of elif section.

if: some condition is True:

do something

elif: some other condition is True

do something different

else:

do something.

We can have multiple elif(s) and else is the end statement that is if none of the conditions match then run this condition.

Single equal sign = means assigning a variable and == means check for a value and if it’s true then do something.

Also Spacing and Capitalization everything matters. Jon Snow != jon Snow

name = "Arya Stark"

if name == "Arya Stark":

print("Valar Morghulis")

The code above prints Valar Mor…

But if you take away an indented space of print like this

name = "Arya Stark"

if name == "Arya Stark":

print("Valar Morghulis")

Then the code above thrown indentation error saying it expects an indented block since it is a subordinate of if block.

Default tab size = 4

Also colons : are really really important.

All statements without colon: will be marked invalid.

**065. Multiple Elifs**

It’s important to have one if. And you can only have one else at the end it’s a simple if nothing then use this block of code kind of thing.

color = input("What is your favorite color ?")

if color == "purple":

print("Excellent Choice!")

elif color == "teal":

print("Not Bad!")

elif color == "pure darkness":

print("I like how you think!")

elif color == 'seafoam':

print('Mediocre')

else:

print("You Monster!")

PS W:\Tony Stark I\CS Resources and Links\Code Advice + College + Notes\CS\Colt Steele Udemy The Modern Python 3 BOOTCAMP\Colt\_Steele\_Python\_BC\ConditionalLogic> python.exe .\logic.py

What is your favorite color ?purple

Excellent Choice!

PS W:\Tony Stark I\CS Resources and Links\Code Advice + College + Notes\CS\Colt Steele Udemy The Modern Python 3 BOOTCAMP\Colt\_Steele\_Python\_BC\ConditionalLogic> python.exe .\logic.py

What is your favorite color ?pure darkness

I like how you think!

**066. A word on truthiness A quick example**

Is is somewhat like == sign where we check for equality sign but it is slightly different.

x = 1

x is 1 #True Truthy expression

x is 0 #False Falsy expression

Things that are inherently by default are false on their own include empty strings. The value none and zero 0 number are also false.

animal = input("Enter your favorite animal: ")

print(f"{animal} is my favorite too!")

PS > python.exe .\truthiness.py

Enter your favorite animal: Black Panther

Black Panther is my favorite too!

PS > python.exe .\truthiness.py

Enter your favorite animal:

is my favorite too!

So what happens here is that a blank string also gets printed out even if we didn’t enter an animal name. As we know empty strings are falsy by default so we will add a check for empty string and if false (which every string is) then we can add an else condition saying you didn’s say anything.

animal = input("Enter your favorite animal: ")

if animal:

print(f"{animal} is my favorite too!")

else:

print("You didn't say anything!")

Now when we encounter empty string we get this

PS > python.exe .\truthiness.py

Enter your favorite animal:

You didn't say anything!

Besides false conditional checks other things that are naturally falsy include empty objects, empty strings, None and zero 0.

**067 The Joy of Comparison Operators**

Here we will see what we can put in if some condition is true part

if: some condition is True:

do something

elif: some other condition is True

do something different

else:

do something.

Comparison operators allows us to compare two pieces of data. a = 1 and b = 1

|  |  |  |
| --- | --- | --- |
| **Op** | **What it does** | **Example** |
| == | Truthy if **a** has the same value as **b** | a == b #True |
| != | Truthy if **a** does **NOT** has the same value as **b** | a != b #False |
| >  < | Truthy if **a** is greater than **b**  Truthy if **a** is less than **b** | a > b #False  a < b #False |
| >=  <= | Truthy if **a** is greater than or equal to **b**  Truthy if **a** is less than or equal to **b** | a >= b #True  a <= b #True |

>>> 1 == 1

True

>>> 1 == 2

False

>>> 1 != 1

False

>>> color = "teal"

>>> color != "purple"

True

>>> 1 < 3

True

>>> -100 > -200

True

>>> "a" < "A"

False

>>> "A" < "a"

True

>>> age = 18

>>> age <= 18

True

>>> if age >= 18:

... print("YOU ARE AN ADULT!")

...

YOU ARE AN ADULT!

**068 Logical Operators AND OR**

These are logical operators and you can throw these in with a conditional operators like if.

They work slightly in a different way and connect other comparison operators slightly together.

Helps us to make more complex logical statements.

|  |  |  |
| --- | --- | --- |
| **Op** | **What It Does** | **Example** |
| And | Truthy if both a and b are true  (logical conjunction) | **if** a **and** b:  print(c) |
| or | Truthy if either of **a** OR **b** are true  (logical disjunction) | **If** am\_tired **or** is\_bedtime:  print(“go to sleep”) |
| Not | Truthy if the opposite of a is true (logical negation) | **if not** is\_weekend:  print(“go to work”) |

**AND**

>>> age = 2

>>>

>>> age = 6

>>> age > 2

True

>>> age < 8

True

>>> age > 2 and age < 8

True

>>> age > 6 and age < 8

False

>>> if age > 2 and age < 8:

... print("YOU PAY CHILD PRICE")

...

YOU PAY CHILD PRICE

**OR**

city = input("Where do you live ? ")

if city == "los angeles" or city == "san francisco":

print("You live in California!")

else:

print("You live somewhere else!")

>>> 1 < 3 or 1 == 99

>>> True #Since it is or only one side true is sufficient to show true.

**70. NOT**

Used to negate other pieces of logic.

>>> age = 10

>>> age < 4

False

>>> not age < 4

True

**71. Is vs. ==**

Is and == both test for equality but it is somehow connected to memory storage in python.

It depends upon what the meaning of word “is” is. – Bill Clinton.

In python “==” and “is” are very similar i.e. comparators. However they aren’t same.

a = 1

a == 1 #True

a is 1 #True

But if you compare two data storage structures like lists you might spot a difference such as:

a = [1, 2, 3] # A list of numbers

b = [1, 2, 3]

a == b #True As the value of the lists are same.

a is b #False But is gives false. Because == checks if values are same or not a is b checks if both a and list b are stored in the memory in the same place.

i.e. To compare the values if they are same data wise two lists or data structures we would use **==**. But if you want to see whether two objects are exactly the same and are stored same place in the memory then we use **is**.

Now if we point to the same list and say now

c = b

b is c #True

== Checks if Value stored is same

is Checks where stored in memory at same place.

72. Bouncer Code-Along and Nested Conditionals

In file comments