**I  
INTRODUCTION**

1. **Course Outline**  
   Python Basics  
   Python Basics 2  
   Developer Environments: Terminal, Code Editors, IDEs, Notebooks  
   Advanced Python  
   Careers  
   Scripting in Python  
   Data Scrapping  
   Automation: Selenium  
   Web Development  
   ML & Data Science
2. **Join Our Online Classroom!**  
   [*ZeroToMastery (discord.com)*](https://discord.com/invite/9KxUUxt7Vd)
3. **Exercise: Meet The Community**

**II  
PYTHON INTRODUCTION**

1. **What Is a Programming Language**Set of instructions written in a high-level language which can be translated to machine language by compiler/interpreter.  
   Assembly language: Close to machine language.  
   Python: Close to English.  
   Compiler/Interpreter: Translator between High level language and Machine language.  
   Compiler: Translates entire code all at once and creates a Binary file.  
   Interpreter: Translates codes one line at a time and creates a Byte code which runs on a Cpython Virtual Machine.
2. **Python Interpreter**Python official website: [*https://www.python.org/*](https://www.python.org/) - cPython  
   Jython (Java), PyPy (Python), IronPython (DotNet)
3. **How To Run Python Code**Online repl: [*https://replit.com/*](https://replit.com/)Online repl: [*https://glot.io/*](https://glot.io/)
4. **Note Jumping To Developer Environment**
5. **Our First Python Program**name = input(‘What is your name?\n’)  
   print(‘Hello ’ + name)
6. **Python 2 Vs Python 3**Python (2) was created by “Guido Van Rossum” in 1991. Name was coined from “Monty Python’s Flying Circus”, a comedy series from the 1970s.  
   In 2008, they created Python 3. Some of the features of Python 2 doesn’t work in Python 3 due to some breaking changes.  
     
   Story of Python by Guido Van Rossum: [*https://www.youtube.com/watch?v=J0Aq44Pze-w*](https://www.youtube.com/watch?v=J0Aq44Pze-w)  
   Python 2 Vs Python 3: <https://sebastianraschka.com/Articles/2014_python_2_3_key_diff.html> Python 2 Vs Python 3: [*https://www.geeksforgeeks.org/important-differences-between-python-2-x-and-python-3-x-with-examples/*](https://www.geeksforgeeks.org/important-differences-between-python-2-x-and-python-3-x-with-examples/)
7. **Why So Many Languages**Every Languages have their own strengths and trade-offs.  
   Some languages are fast in executing but slow in writing, like C/C++. On the other hand some are fast in writing and slow in executing, like Python.
8. **Exercise How Does Python Work**  
   Teach someone how fundamentals of Python, how it works.
9. **ZTM Python Cheat Sheet**Python cheat-sheet: [*https://github.com/aneagoie/ztm-python-cheat-sheet*](https://github.com/aneagoie/ztm-python-cheat-sheet)

**III  
PYTHON BASICS**

1. **Learning Python**Terms  
   Data Types  
   Actions  
   Best Practices
2. **Python Data Types**A data type is a classification that specifies which type of value a variable has and what type of mathematical, relational or logical operations can be applied to it without causing an error.  
     
   ***Fundamental Data Types***int  
   float  
   bool  
   str  
   list  
   tuple  
   set  
   dict ***Classes:*** Custom Types ***Modules:*** Specialized Data TypesNone
3. **Latest Version of Python**Python official website: [*https://www.python.org/*](https://www.python.org/)
4. **How to Succeed**Code Alongside the course.
5. **Numbers**Integers & floating point numbers  
   # int & float  
   \*\* -to the power of  
   // -integer division rounded down  
   % -modulo/remainder  
     
   Floating point numbers: [*https://www.youtube.com/watch?v=PZRI1IfStY0*](https://www.youtube.com/watch?v=PZRI1IfStY0)
6. **Math Functions**round : round to nearest integer  
   print(round(3.9)) #4  
   abs : Absolute value  
   print(abs(-20)) #20  
     
   Mathematical functions:[*https://docs.python.org/3/library/math.html*](https://docs.python.org/3/library/math.html)Mathematical functions: [*https://www.programiz.com/python-programming/modules/math*](https://www.programiz.com/python-programming/modules/math)
7. **Developer Fundamentals I  
     
   *Don’t read the dictionary:***   
   Don’t memorize every single thing, just know things are there. Focus on using language.
8. **Operator Precedence**()  
   \*\*  
   \* /  
   + -
9. **Exercise Operator Precedence**Operator Precedence Exercise: [*https://replit.com/@aneagoie/Operator-Precedence*](https://replit.com/@aneagoie/Operator-Precedence)
10. **Optional: bin() and complex**complex: real number + imaginary number  
    Python int function: [*https://www.geeksforgeeks.org/python-int-function/*](https://www.geeksforgeeks.org/python-int-function/)   
    bin(): Binary number  
    print(bin(5)) #0b101  
    print(int(‘0b101’, 2)) #5 -(“number”, base of number)
11. **Variables**A variable is a value that can change depending on conditions or on information passed to the program.snake\_case  
    Starts with lower case or underscore  
    Letters, numbers or underscores  
    Case sensitive  
    Don’t overwrite keywords  
    Variable names should be descriptive  
      
    Variables in python starting with underscore ( \_ ) signifies private variables.  
    Python Keywords: [*https://www.w3schools.com/python/python\_ref\_keywords.asp*](https://www.w3schools.com/python/python_ref_keywords.asp)   
      
    ***Constants:***  
    Keep constants capital.  
    PI = 3.14  
    Values of constants should not be changed once assigned.   
      
    ***Dunder Variables:***  
    Starts with two underscores.  
    We should not create variables starting with two underscores.  
      
    a, b, c = 1, 2, 3
12. **Expressions vs Statements**iq = 100  
    user\_age = iq / 5  
      
    An expression is a piece of code which produces a value, (iq/5) is an expression here. An entire line of code is a statement.
13. **Augmented Assignment Operator**some\_value = 5  
    some\_value += 2 # some\_value = some\_value + 2  
    Augmented Assignment Operator repl: [*https://replit.com/@aneagoie/Augmented-Assignment-Operator*](https://replit.com/@aneagoie/Augmented-Assignment-Operator)
14. **Strings**Strings can be denoted using single quotes or double quotes and triple quotes for multi-line strings.
15. **String Concatenation**it means adding strings together.  
    print(‘hello’ + ‘Nitesh’)  
    print(‘hello’ + 5) #error
16. **Type Conversion**print(str(100))  
    print(type(str(100)))  
    print(type(int(str(100))))  
    print("Hello " + str(21))  
    a = str(100)  
    b = int(a)  
    c = type(b)  
    print(c)
17. **Escape Sequences**weather = “\t It\’s \“kind of\” sunny \n Hope you have a good day!”  
    print(weather)  
      
    ‘\t’: tab  
    ‘\n’: new line
18. **Formatted Strings**name = "Nitesh"

age = 27

print("Hi " + name + "! You are " + str(age) + " years old.\n")

# Python 3 formatted strings

print(f"Hi {name}! You are {age} years old.\n")  
  
String Formatting: <https://replit.com/@aneagoie/string-formatting>

1. **String Indexes**String Slicing**:** [start: stop: step-over]selfish = ‘01234567’  
    #01234567  
     
   print(selfish[::-1]) # Reverses the string  
   String Indexes Exercise: [*https://replit.com/@aneagoie/string-indexes*](https://replit.com/@aneagoie/string-indexes)
2. **Immutability**Strings in python are immutable, means it cannot be changed.  
   Part of a string can’t be assigned, although whole string can be reassigned a new value.
3. **Built in Functions and Methods**Python Built-in Functions: [*https://docs.python.org/3/library/functions.html*](https://docs.python.org/3/library/functions.html)Python String Methods: [*Python String Methods (w3schools.com)*](https://www.w3schools.com/python/python_ref_string.asp)  
   Methods are actions which are owned by the objects, like strings or lists etc.  
   Python string methods doesn’t replace the original string.
4. **Booleans**bool = True/False  
   True = 1  
   False = 0
5. **Exercise: Type Conversion**input() function returns a string.  
   Q.> Input “What year were you born?” and print the age.
6. **Developers Fundamentals: II**Comments: Python uses # to add comment in code.  
   Comments should be added when something complex is written in code to let other programmers understand the piece of code.  
   Comments should be concise and minimal.  
     
   Writing Comments in Python (Guide): [*https://realpython.com/python-comments-guide/*](https://realpython.com/python-comments-guide/)
7. **Exercise: Password Checker**Q.> Input username and password and print “Password is --- length long”.
8. **Lists**List is an ordered sequence of objects.List items are contained within square brackets [].
9. **List Slicing**list[start : stop : stepover]  
   Lists are mutable, that is list items can be changed/updated.  
   List slicing doesn’t change the original list but creates a new list.  
     
   Lists Exercise: [*https://replit.com/@aneagoie/lists*](https://replit.com/@aneagoie/lists)
10. **Matrix**Matrix is a multi-dimensional list/array.  
    Matrix Exercise: [*https://replit.com/@aneagoie/matrix*](https://replit.com/@aneagoie/matrix)
11. **List Methods**Append/Insert/Extend/Remove changes the list in-place. It doesn’t create a new list.  
    Append adds item to the end of the existing list.  
    Insert adds item to the desired index.  
    Extend adds items (iterable) at the end of the list.  
    Pop removes the last element from the list. Pop(index) removes the element at the index in the list.  
    Pop returns the popped element from the list.  
    Remove(value) removes the given value from the list.  
    Clear removes all the elements of the list.  
    Python List Methods: [*https://www.w3schools.com/python/python\_ref\_list.asp*](https://www.w3schools.com/python/python_ref_list.asp)
12. **List Methods 2**Index returns the first occurrence for the element in the list.  
    In keyword returns if some element/string is present in the list/string (True/False).  
    Count returns the number of times the element occurs in the list.  
    Python Keywords: [*https://www.w3schools.com/python/python\_ref\_keywords.asp*](https://www.w3schools.com/python/python_ref_keywords.asp) List Methods Exercise: [*https://replit.com/@aneagoie/lists-2*](https://replit.com/@aneagoie/lists-2)
13. **List Methods 3**Sort method sorts the list in-place.  
    Sorted() function sorts the list out-of-place. It doesn’t change the original list.  
    Copy method copies the list and returns the same.  
    Reverse method reverses the list. It doesn’t sort.
14. **Common List Patterns**print(basket[::-1) #Reverses the list, out-of-place  
    List Methods Exercise 2: [*https://replit.com/@aneagoie/lists-3*](https://replit.com/@aneagoie/lists-3)
15. **List Unpacking**

a, b, c, \*other, d = [1, 2, 3, 4, 5, 6, 7, 8, 9]

a= 1, b= 2, c= 3, other= [4, 5, 6, 7, 8], d= 9

1. **None**None is a datatype in Python. It refers to absence of value. In other programming languages it is denoted by Null.
2. **Dictionaries**A dictionary is an unordered Key-Value pair.  
   dictionary = {key: value}
3. **Developer Fundamentals III**Understanding Data Structures.  
   When to use which data structure.
4. **Dictionary Keys**Dictionary keys must be immutable like strings, integers, booleans etc.  
   Dictionary key needs to be unique otherwise its value gets overwritten.
5. **Dictionary Methods**.get() method returns the value of the argument key. It returns None/default value if key doesn’t exist is the dictionary.  
   Python Dictionary Methods: [*https://www.w3schools.com/python/python\_ref\_dictionary.asp*](https://www.w3schools.com/python/python_ref_dictionary.asp)
6. **Dictionary Methods 2**Dictionary Exercise:[*https://replit.com/@aneagoie/dictionary*](https://replit.com/@aneagoie/dictionary)
7. **Tuples**Tuples are immutable lists.  
   Tuple is faster than list as data in tuples cannot be changed.
8. **Tuples 2**Tuples with single item in it has a comma (,) attached to its only value. Eg., (2,)  
   Tuples have only two methods: count() and index()  
   Python Tuple Methods: [*https://www.w3schools.com/python/python\_ref\_tuple.asp*](https://www.w3schools.com/python/python_ref_tuple.asp)
9. **Sets**

Sets are unordered collection of unique objects.  
In Sets, values are wrapped around curly brackets.

1. **Sets 2**Python Set Methods: [*https://www.w3schools.com/python/python\_ref\_set.asp*](https://www.w3schools.com/python/python_ref_set.asp)   
     
   .difference()  
   .discard()  
   .difference\_update()  
   .intersection()  
   .isdisjoint()  
   .issubset()  
   .issuperset()  
   .union()  
     
   Sets Exercise: [*https://replit.com/@aneagoie/sets*](https://replit.com/@aneagoie/sets)

**IV  
Python Basics II**

1. **Breaking the Flow**Conditions, conditional logics, loops, looping.
2. **Conditional Logic**if condition:  
   statement  
   elif:  
    statement  
   else:  
    statement  
     
   else block runs only when all if and elif conditions evaluates to false.
3. **Indentation in Python**In Python, indentation is important for a block of code.  
   Python follows 4 spaces by default for indentation.
4. **Truthy vs Falsy**Python converts conditions to Boolean inclusively.  
   What is Truthy and Falsy: [*https://stackoverflow.com/questions/39983695/what-is-truthy-and-falsy-how-is-it-different-from-true-and-false*](https://stackoverflow.com/questions/39983695/what-is-truthy-and-falsy-how-is-it-different-from-true-and-false)
5. **Ternary Operator**Ternary Operators are also called as Conditional Expressions.  
   Statement 1 if condition else Statement 2
6. **Short Circuiting**If two (or more) conditions are there, with OR keyword and if first condition is TRUE, then second condition is (consecutive conditions are) ignored and python interpreter goes directly to next statement.  
   If two (or more) conditions are there, with AND keyword and if first condition is FALSE, then second condition is (consecutive conditions are) ignored and python interpreter goes directly to next statement.  
   These situations are called as short circuiting.
7. **Logical Operators  
     
   >  
   <  
   ==  
   >=  
   <=  
   !=**  
   or  
   and  
   not
8. **Exercise Logical Operators**is\_magician = False  
   is\_expert = True  
     
   # Check if magician and expert: "you are a master magician"  
   # Check if magician but not expert: "at least you're getting there"  
   # Check if not a magician: "You need magic powers”
9. **Is vs ==**  
   “**==**” checks for equality of value, while “is” checks for equality of memory locations.  
   “is” is stricter than “**==**”
10. **For Loops**  
    Syntax: for variable in iterable:   
     statements
11. **Iterables**  
    Iterable is an object or a collection of objects which can be iterated over.  
    Iterable can be a list, dictionary, tuple, set, string.
12. **Exercise Tricky Counter**# Write a program to find the sum of items in the listTricky Counter: [*https://replit.com/@aneagoie/loops#main.py*](https://replit.com/@aneagoie/loops#main.py)
13. **Range()**  
    Range() is an object that produces a sequence of integers from start (inclusive) to stop (exclusive).  
    Syntax: range(start, stop, step-over)  
    In for loop, if we don’t have to use variable inside the loop, we can simply put “\_” in place of variable.  
    Syntax: for \_ in range(100):  
     statements
14. **Enumerate()**  
    The enumerate object yields pairs containing an index (from start, which by default starts with zero) and a value yielded by the iterable argument.
15. **While Loops**  
    Syntax: while condition:  
     statements  
     else:  
     statements  
    Else block will only execute when while condition turns False. Else block will not execute if there is a break statement in while block.
16. **While Loops 2**  
      
    while True:  
     statements  
     condition: break  
      
    While loops are more useful when we don’t know how many times the loop is going to run.
17. **Break, Continue, Pass**  
    **Break** stops the loop ignoring all consequent iterations.  
    **Continue** skips to next iteration of the loop.  
    **Pass** does nothing but passes to next line. Pass is used as placeholder, to put piece of code later.
18. **Our First GUI**  
    # Display the image below to the right hand side where the 0 is going to be ' ',  
    # and the 1 is going to be '\*'. This will reveal an image!

picture = [  
 [0, 0, 0, 1, 0, 0, 0],  
 [0, 0, 1, 1, 1, 0, 0],  
 [0, 1, 1, 1, 1, 1, 0],  
 [1, 1, 1, 1, 1, 1, 1],  
 [0, 0, 0, 1, 0, 0, 0],  
 [0, 0, 0, 1, 0, 0, 0]  
]

1. **Developer Fundamentals IV**  
   Clean  
   Readability  
   Predictability  
   DRY: Do Not Repeat yourself
2. **Exercise Find Duplicates**  
   # Check for duplicates in the list.  
   # Print the characters which have duplicates in the list.  
     
   some\_list = ['a', 'b', 'c', 'b', 'd', 'm', 'n', 'n']  
     
   Find Duplicates: [*https://replit.com/@aneagoie/Find-Duplicates*](https://replit.com/@aneagoie/Find-Duplicates)
3. **Functions**  
   Syntax: def function\_name():  
    statements  
     
   Function naming convention is same as variables in Python.  
   Before calling a function, we must define the function.
4. **Parameters and Arguments**  
   Parameters are the values received by the function when we define the function.  
   Arguments are the values we provide to a function while calling/invoking.
5. **Default Parameters and Keyword Arguments**  
   Default parameters makes sure function gets the required parameters even if function is called wrong way.  
     
   Keyword arguments removes the restriction of position of arguments while calling the function. Keyword arguments increases the readability of code. While using keyword arguments, order of the arguments does not matter.  
   If we don’t use keyword arguments during calling the function, then the arguments will be positional, called as positional arguments.
6. **Return**  
   Functions have to return something whenever it is called. If no return keyword is in function definition then it returns **None**.   
     
   A function should do one thing very well.  
   A function should return something.  
     
   Return keyword exits the function immediately.
7. **Exercise Tesla**  
   Exercise Tesla: [*https://replit.com/@aneagoie/tesla*](https://replit.com/@aneagoie/tesla)   
   Solution Tesla: [*https://replit.com/@aneagoie/tesla-1*](https://replit.com/@aneagoie/tesla-1)
8. **Methods Vs Functions**  
   Methods have to be owned by some object or data type.  
   Functions can be called with arguments of required types of parameters.  
   Both methods and functions allows us to take actions on our data types.
9. **Docstrings**  
   At the beginning of the function, we can add docstring to add info to the function.  
   Docstring is added within triple double-quotes.  
     
   # Syntax  
   def <function\_name>(a):  
    *"""  
    Info : This is a function which takes param a  
    :param a:  
    :return:  
    """* <function statements here>
10. **Clean Code**  
    # Write a function to check if number is even or not.
11. **Args and Kwargs**  
    With \*args and \*\*kwargs, we can take multiple arguments as parameters in a function.  
    Python stores \*args as a tuple and \*\*kwargs as a dictionary in memory. We can give as many arguments as possible using \*args and \*\*kwargs.  
    Rule for ordering of parameters: params, \*args, default parameters, \*\*kwargs.
12. **Exercise Functions**# Highest Even: Write a function to find the highest even number from the list.  
    Solution: [*https://replit.com/@aneagoie/highest-even-solution*](https://replit.com/@aneagoie/highest-even-solution)
13. **Scope**  
    Scope means – What variables do I have access to?  
    Variables in Python have functional scope.  
    If a variable is tried to access outside its scope, then python interpreter will throw a name error.
14. **Scope Rules**  
    #1 - Start with local  
    #2 - Parent local  
    #3 - Global  
    #4 - Built in python functions
15. **Global Keyword**  
    Function parameters are considered as local variables.  
    Global keyword is used to access global variables inside a function.
16. **Nonlocal Keyword**  
    Nonlocal keyword is used to refer to parent function’s variables.  
    Nonlocal keyword: [*https://replit.com/@aneagoie/nonlocal#main.pyhttps://replit.com/@aneagoie/nonlocal#main.py*](https://replit.com/@aneagoie/nonlocal#main.pyhttps://replit.com/@aneagoie/nonlocal)
17. **Why Do We Need Scope**  
    Scopes are used because our system don’t have infinite memory.  
    After a function has finished executing, python interpreter runs garbage collector and removes all the variables and actions inside it.
18. **Python Exam Testing Your Understanding**  
    Python Quiz: [*https://www.w3schools.com/quiztest/quiztest.asp?qtest=PYTHON*](https://www.w3schools.com/quiztest/quiztest.asp?qtest=PYTHON)   
    Python Exercise: [*https://www.w3schools.com/python/exercise.asp*](https://www.w3schools.com/python/exercise.asp)

**V  
Developer Environment**

1. **MAC – Python Install**  
   Python installation guide for MAC systems: [*https://osxdaily.com/2018/06/13/how-install-update-python-3x-mac/*](https://osxdaily.com/2018/06/13/how-install-update-python-3x-mac/)
2. **Windows – Python Install**Python official website: [*https://www.python.org/*](https://www.python.org/)Python installation guide for Windows systems: [*https://realpython.com/installing-python/*](https://realpython.com/installing-python/)
3. **Windows – Get Terminal**  
   Terminal installation guide for Windows systems: [*https://www.thewindowsclub.com/install-windows-terminal-on-windows-10*](https://www.thewindowsclub.com/install-windows-terminal-on-windows-10)
4. **Linux – Python Install**  
   Python installation guide for Linux systems: [*https://realpython.com/installing-python/*](https://realpython.com/installing-python/)
5. **Python Developer tools**  
   Terminal  
   Code Editors – lightweight  
   IDEs – Full-fledged environment to write and run code  
   Jupyter Notebooks
6. **Sublime Text**Sublime Text official website: [*https://www.sublimetext.com/3*](https://www.sublimetext.com/3)
7. **Quick Note Upcoming Video**[*The Complete Web Developer in 2023: Zero to Mastery | Udemy*](https://www.udemy.com/course/the-complete-web-developer-zero-to-mastery/?referralCode=FFF295AECF3594CE440E)
8. **Optional Terminal Commands**ls #list // lists all files and folders in directory

pwd # present working directory

cd <folder name> #move pwd to argument folder

cd .. #one directory up

cd / #move to root directory

cd ~ #move to user directory  
clear #clears the terminal

open . #opens current directory (LINUX)  
open <file name> #opens argument file (LINUX)

open –a “<application name>” #open specified application

open –a “<application name>” <file name> # opens specified application with specified application  
start . #opens current directory (WINDOWS)

start <file name> #opens argument file (WINDOWS)

mkdir <folder name> #creates new folder of argument name

touch <file name> #creates file of argument name

mv <old file name> <new file name> #rename file name

rm <file name> #delete argument file  
rm –r <folder name> #delete argument folder (LINUX)  
rmdir <folder name> # delete argument folder (WINDOWS)  
  
TAB key: Auto fill  
UP ARROW key: command history

1. **iTerm Setup**
2. **Using the command line terminal**  
   python <python file> #runs file with program in it
3. **Visual Studio Code**  
   Visual Studio Code official website: [*https://code.visualstudio.com/downloads*](https://code.visualstudio.com/download)
4. **PyCharm**Jetbrains official website: [*https://www.jetbrains.com/pycharm/download/#section=windows*](https://www.jetbrains.com/pycharm/download/#section=windows)
5. **Code Formatting – PEP 8**  
   PEP stands for Python Enhancement Proposals.  
   There are lots of PEPs.  
   PEP 8 is style guide for Python code.
6. **Quick Note Jupyter Notebooks**
7. **Jupyter Notebooks**  
   Anaconda Official Website: [*https://problemsolvingwithpython.com/01-Orientation/01.03-Installing-Anaconda-on-Windows/*](https://problemsolvingwithpython.com/01-Orientation/01.03-Installing-Anaconda-on-Windows/)   
   Jupyter Notebook uses browser as GUI. It doesn’t have its own code editor.  
   Jupyter runs program line by line as soon as it is written. It auto saves file.
8. **Section Review**

**VI  
Advanced Python Object Oriented Programming**

1. **Note On Upcoming Sections  
     
   Ps: I will be using tools like PyCharm and Jupyter Notebooks in the sections where we start building our 12+ projects later in the course... so don't worry :)**
2. **What is OOP**  
   Object Oriented Programming is a programming paradigm.  
   Everything in Python is an Object.
3. **What is OOP part 2**[*History of programming languages - Wikipedia*](https://en.wikipedia.org/wiki/History_of_programming_languages)Class names should be Camel Case.  
   Class is a blueprint for objects. Object is instance of a class.  
   Creating a new object is instantiating a class.  
     
     
   class BigObject:  
    pass  
     
     
   obj1 = BigObject()  
   obj2 = BigObject()  
   obj3 = BigObject()  
     
   print(type(obj1))
4. **Creating Our Own Objects**  
   \_\_init\_\_ is a constructor method. This is automatically called when class is instantiated.  
   While defining a method in a class, first parameter is **self**.  
   Different parameters of constructor method are called as attributes of the class.  
     
     
   class PlayerCharacter:  
    def \_\_init\_\_(self, name, age):  
    self.name = name  
    self.age = age  
     
    def run(self):  
    print('Run')  
     
     
   player1 = PlayerCharacter('Cindy', 44)  
   player2 = PlayerCharacter('Tom', 21)  
   player2.attack = 50  
     
   print(player1.name)  
   print(player2.name)  
   print(player1.age)  
   print(player2.age)  
   player1.run()  
   print(player2.attack)
5. **Attributes and Methods**  
   Attributes are piece of data which is dynamic in class, specific to objects.  
   Class Object Attribute is static in class. It doesn’t change with its instances.  
   Class Object Attribute is defined outside the methods.  
     
   class PlayerCharacter:  
    # Class Object Attribute  
    membership = True  
     
    def \_\_init\_\_(self, name, age):  
    if self.membership: # Or PLayerCharacter.membership  
    self.name = name  
    self.age = age  
     
    def shout(self):  
    print(f'My name is {self.name}')  
     
    def run(self, hello):  
    print(f'{hello} {self.name}')  
     
     
   player1 = PlayerCharacter('Cindy', 44)  
   player1.shout()  
   player1.run('Hi')  
   print(player1.membership, '\t', player1.name, '\t', player1.age)
6. **\_\_init\_\_**  
   class PlayerCharacter:  
    # Class Object Attribute  
    membership = True  
     
    def \_\_init\_\_(self, name, age):  
    if age > 18:  
    self.name = name  
    self.age = age  
     
    def shout(self):  
    print(f'My name is {self.name}')  
     
     
   player1 = PlayerCharacter('Tom', 20)  
   print(player1.shout())
7. **Exercise Cats Everywhere**  
   # Given the below class:  
     
   class Cat:  
    species = 'mammal'  
     
    def \_\_init\_\_(self, name, age):  
    self.name = name  
    self.age = age  
     
     
   # 1 Instantiate the Cat object with 3 cats.  
   cat1 = Cat('cat1', 5)  
   cat2 = Cat('Cat2', 7)  
   cat3 = Cat('Cat3', 3)  
     
     
   # 2 Create a function that finds the oldest cat.  
   def oldest\_cat(\*args):  
    return max(args)  
     
     
   # 3 Print out: "The oldest cat is x years old.".  
   # x will be the oldest cat age by using the function in #2  
   print(f'Oldest Cat is {oldest\_cat(cat1.age, cat2.age, cat3.age)} years old.')
8. **@classmethod and @staticmethod**  
   @classmethod is a method on actual class. We don’t need to instantiate class to run classmethod. In classmethods, we have access to class attributes.  
   @staticmethod is also a method on actual class. We don’t have access to cls in staticmethod.  
   [*Instance vs. Static vs. Class Methods in Python: The Important Differences (makeuseof.com)*](https://www.makeuseof.com/tag/python-instance-static-class-methods/)  
     
     
   class PlayerCharacter:  
    def \_\_init\_\_(self, name, age):  
    self.name = name  
    self.age = age  
     
    @classmethod  
    def adding\_things(cls, num1, num2):  
    return cls('Teddy', num1 + num2)  
     
    @staticmethod  
    def adding\_things2(num1, num2):  
    return num1 + num2  
     
     
   player3 = PlayerCharacter.adding\_things(2, 3) # Instantiated class using classmethod  
   print(player3)  
   print(PlayerCharacter.adding\_things2(5, 9))
9. **Reviewing What We Know So Far**  
   Class, Object, Attributes, Methods, @classmethod, @staticmethod.
10. **Developer Fundamental V**  
    Test your assumptions.  
      
    help(object)
11. **Encapsulation**  
    Encapsulation is binding of data and operations that manipulate that data.  
    Data = Attributes, Operations = Methods  
      
    class PlayerCharacter:  
     def \_\_init\_\_(self, name, age):  
     if age > 18:  
     self.name = name  
     self.age = age  
      
     def speak(self):  
     print(f'My name is {self.name} and I am {self.age} years old.')  
      
      
    player1 = PlayerCharacter('Tom', 20)  
    player1.speak()
12. **Abstraction**  
    Abstraction means hiding information and giving access only to what is necessary.  
    We don’t need to know how the data is grabbed while using methods.
13. **Private Vs Public Variables**  
    In Python, there is no true private variable.  
    A variable starting with “\_” (underscore) is a private variable in python and we should not overwrite it.  
    We should not write variable names starting with two underscore. Keywords starting with two underscore are Dunder methods.
14. **Inheritance**  
    Inheritance allows new objects to take properties of old objects.  
      
    If we don’t need a variable or attributes to be used then we don’t need to explicitly define \_\_init\_\_ method.  
      
    # Parent Class  
    class User:  
     def sign\_in(self):  
     print('logged in')  
      
      
    # Sub Class/ Child Class/ Derived Class  
    class Wizard(User):  
     def \_\_init\_\_(self, name, power):  
     self.name = name  
     self.power = power  
      
     def attack(self):  
     print(f'Attacking with power of {self.power}')  
      
      
    class Archer(User):  
     def \_\_init\_\_(self, name, num\_arrows):  
     self.name = name  
     self.num\_arrows = num\_arrows  
      
     def attack(self):  
     print(f'Attacking with arrows: Arrows left- {self.num\_arrows}')  
      
      
    wizard1 = Wizard('Merlin', 50)  
    archer1 = Archer('Robbin', 100)  
    wizard1.attack()  
    archer1.attack()
15. **Inheritance 2**  
    isinstance() is a built-in function in python to check if something is instance of some class.  
      
    print(isinstance(wizard1, Wizard))  
    print(isinstance(wizard1, User))  
    print(isinstance(wizard1, object))
16. **Polymorphism**  
    Poly means “many” and morphism means “forms”. Polymorphism allows us to have many forms of same object.  
      
    # Parent Class  
    class User:  
     def sign\_in(self):  
     print('logged in')  
      
     def attack(self):  
     print('Do nothing.')  
      
      
    # Sub Class/ Child Class/ Derived Class  
    class Wizard(User):  
     def \_\_init\_\_(self, name, power):  
     self.name = name  
     self.power = power  
      
     def attack(self):  
     print(f'Attacking with power of {self.power}')  
      
      
    class Archer(User):  
     def \_\_init\_\_(self, name, num\_arrows):  
     self.name = name  
     self.num\_arrows = num\_arrows  
      
     def attack(self):  
     print(f'Attacking with arrows: Arrows left- {self.num\_arrows}')  
      
      
    wizard1 = Wizard('Merlin', 50)  
    archer1 = Archer('Robbin', 30)  
      
      
    def player\_attack(char\_):  
     char\_.attack()  
      
      
    player\_attack(wizard1)  
    player\_attack(archer1)  
      
    for char in [wizard1, archer1]:  
     char.attack()
17. **Exercise Pets Everywhere**  
    class Pets:  
     animals = []  
      
     def \_\_init\_\_(self, animals):  
     self.animals = animals  
      
     def walk(self):  
     for animal in self.animals:  
     print(animal.walk())  
      
    class Cat:  
     is\_lazy = True  
      
     def \_\_init\_\_(self, name, age):  
     self.name = name  
     self.age = age  
      
     def walk(self):  
     return f'{self.name} is just walking around'  
      
    class Simon(Cat):  
     def sing(self, sounds):  
     return f'{sounds}'  
      
    class Sally(Cat):  
     def sing(self, sounds):  
     return f'{sounds}'  
      
    # 1 Add another Cat  
    class Chilli(Cat):  
     def sing(self, sounds):  
     return f'{sounds}'  
      
    # 2 Create a list of all the pets (create 3 cat instances from the above).  
    my\_cats = [Simon(Simon, 5), Sally(Sally, 7), Chilli(Chilli, 3)]  
      
    # 3 Instantiate the Pet class with all your cats use variable my\_pets.  
    my\_pets = Pets(my\_cats)  
      
    # 4 Output all the cats walking using the my\_pets instance.  
    my\_pets.walk()
18. **Super()**  
    super() refers to class above current class.  
      
    class User:  
     def \_\_init\_\_(self, email):  
     self.email = email  
      
     def sign\_in(self):  
     print('logged in')  
      
      
    # Sub Class/ Child Class/ Derived Class  
    class Wizard(User):  
     def \_\_init\_\_(self, name, power, email):  
     super().\_\_init\_\_(email)  
     # same as:: User.\_\_init\_\_(self, email)  
     self.name = name  
     self.power = power  
      
      
    wizard1 = Wizard('Merlin', 60, 'merlin@gmail.com')  
    print(wizard1.email)
19. **Object Introspection**  
    Introspection in computer programming is to determine the type of object at runtime.  
      
    wizard1 = Wizard('Merlin', 60, 'merlin@gmail.com')  
    print(dir(wizard1))
20. **Dunder Methods**  
    Dunder methods are special methods which are inherited to all the objects we create.We can modify dunder methods to add special functionality to our classes.  
      
    [*Data model — Python 3.11.3 documentation*](https://docs.python.org/3/reference/datamodel.html#special-method-names)  
      
    class Toy:  
     def \_\_init\_\_(self, color, age):  
     self.color = color  
     self.age = age  
     self.my\_dict= {  
     'name': 'Yoyo',  
     'has\_pets': False  
     }  
      
     def \_\_str\_\_(self):  
     return f'{self.color}'  
      
     def \_\_len\_\_(self):  
     return 5  
      
     # def \_\_del\_\_(self):  
     # print('deleted!')  
      
     def \_\_call\_\_(self):  
     return 'Yess?'  
      
     def \_\_getitem\_\_(self, i):  
     return self.my\_dict[i]  
      
      
    action\_figure = Toy('red', 0)  
    print(action\_figure.\_\_str\_\_())  
    print(str(action\_figure))  
    print(len(action\_figure))  
    # del action\_figure  
    print(action\_figure())  
    print(action\_figure['name'])

# Exercise**:** By reading the python documentation, add 3 more magic/dunder methods of your choice to this Toy class.def \_\_abs\_\_(self, num):  
 return num  
  
def \_\_hex\_\_(self):  
 return 6  
  
def \_\_set\_\_(self):  
 return 'done setting'  
  
  
print(action\_figure.\_\_abs\_\_(-50))  
print(action\_figure.\_\_hex\_\_())  
print(action\_figure.\_\_set\_\_())

1. **Exercise Extending List**  
   class SuperList(list):  
     
    def \_\_len\_\_(self):  
    return 1000  
     
     
   super\_list1 = SuperList()  
   print(len(super\_list1))  
     
   super\_list1.append(5)  
   print(super\_list1[0])  
   print(issubclass(SuperList, list))  
   print(issubclass(list, object))
2. **Multiple Inheritance**  
   class User:  
    def sign\_in(self):  
    print('logged in')  
     
     
   class Wizard(User):  
    def \_\_init\_\_(self, name, power):  
    self.name = name  
    self.power = power  
     
    def attack(self):  
    print(f'Attacking with power of {self.power}')  
     
     
   class Archer(User):  
    def \_\_init\_\_(self, name, num\_arrows):  
    self.name = name  
    self.num\_arrows = num\_arrows  
     
    def check\_arrows(self):  
    print(f'{self.num\_arrows} remaining')  
     
    def run(self):  
    print('Ran really fast.')  
     
     
   class HybridBorg(Wizard, Archer):  
    def \_\_init\_\_(self, name, power, arrows):  
    Archer.\_\_init\_\_(self, name, arrows)  
    Wizard.\_\_init\_\_(self, name, power)  
     
     
   hb1 = HybridBorg('Borgie', 50, 100)  
   print(hb1.run())  
   print(hb1.check\_arrows())  
   hb1.attack()
3. **MRO – Method Resolution Order**  
   MRO is a rule that python follows to determine which method to run.  
   MRO uses DFS- Depth First Search algorithm.  
     
   [*Method Resolution Order (MRO) in Python (srikanthtechnologies.com)*](http://www.srikanthtechnologies.com/blog/python/mro.aspx)  
     
   class A:  
    num = 10  
     
   class B(A):  
    pass  
     
   class C(A):  
    num = 1  
     
   class D(B, C):  
    pass  
     
   print(D.num)  
   print(D.mro())  
     
     
   class X:  
    pass  
     
   class Y:  
    pass  
     
   class Z:  
    pass  
     
   class A(X, Y):  
    pass  
     
   class B(Y, Z):  
    pass  
     
   class M(B, A, Z):  
    pass  
     
     
   print(M.mro())

**XI  
Modules in Python**

* + - 1. **Modules In Python**  
         Each file of Python in a project is called as module.  
         Naming convention of module is same as variables.  
         import <module name> # import syntax  
         We can import as many modules as required in our project.  
         While importing a module in any project, we don’t need to mention file extension (.py) as all the modules has to be python files.  
         Python interpreter creates a cache and stores a compiled versions of modules to make the code run faster from second time onwards main function is called. Cached version of modules updates every time we change the content of the module.  
         We can access all the functionalities defined in a module through our main function.
      2. **Packages in Python**  
         A folder containing a collection of modules is called as package.  
         import <package name>.<module name> # import syntax  
           
         Python packages have an \_\_init\_\_.py file in it.
      3. **Different Ways To Import**  
           
         import <path to module.module\_name>  
         from <path to module.module\_name> import <function\_name>  
         from <path to module> import <module\_name>  
         from <path to module> import \*
      4. **\_\_name\_\_**All the modules of python are named with their file name.  
         Driver module is named \_\_main\_\_ even if we change the name.  
         if \_\_name\_\_ == ‘\_\_main\_\_’:  
         This syntax is used to ensure the code enclosed within it runs only when the module is explicitly called and not when the module is imported in some other driver module.
      5. **Python Built-in Modules**  
           
         Built-in modules of Python (Standard Library): [*https://docs.python.org/3/py-modindex.html*](https://docs.python.org/3/py-modindex.html)   
         help(<module\_name>) # Documentation of the module  
         dir(<module\_name>) # List all the functions available in the module
      6. **Python Built-in Modules 2**sys.argv # Used to enable users to give arguments directly through terminal.
      7. **Exercise Guessing Game**  
         Solution to Exercise: <https://replit.com/@aneagoie/Guessing-Game>
      8. **Python Package Index**Python Package Index (pypi): [*https://pypi.org/*](https://pypi.org/)   
         Anyone can upload packages in pypi.
      9. **Pip Install**pip –V  
         pip install <pip name> # To install pip from Pypi  
         pip list # lists all the pip installed in the environment  
         Installing pip on all systems: [*https://www.makeuseof.com/tag/install-pip-for-python/*](https://www.makeuseof.com/tag/install-pip-for-python/)
      10. **Virtual Environments**Versioning 1.2.3  
          # 3= bug fixes  
          # 2= new release/added features  
          # 1= breaking changes/major versions.  
            
          We can have different versions of same pip and we can use different ones in different project by installing them separately inside virtual environment.
      11. **Useful Modules**collections # Counter, defaultdict, OrderedDict  
          Counter # Returns the count of every element in the list/str  
          defaultdict # Returns a default value if key doesn’t exist in the dictionary.  
          OrderedDict # Retains the order in which key:value pairs are inserted.
      12. **Useful Modules 2**  
          datetime # Allows to manipulate date and time  
          array # static list in python  
          Array uses less memory and is faster compared to lists.  
            
          When to use array: [*https://stackoverflow.com/questions/176011/python-list-vs-array-when-to-use*](https://stackoverflow.com/questions/176011/python-list-vs-array-when-to-use)
      13. **Developer Fundamentals VI**  
            
          Pros and Cons of libraries:   
          Some libraries might be buggy.  
          We need to use libraries which are maintained well.

**XII  
Debugging In Python**

**How to Debug Code**  
  
Linting: detects errors when we are writing the code.  
Using IDEs and Code Editors: allows us to highlight errors in code.  
  
pdb # Python built-in debugger  
help  
list  
step  
continue  
exit  
  
The Python Debugger: [*https://docs.python.org/3/library/pdb.html*](https://docs.python.org/3/library/pdb.html)

**XIII  
File IO**

* + - 1. **Working With Files In Python**IO stands for input-output.  
         open(<file\_name>): to open any file.  
         .read(): to read entire file. It brings the cursor to the end of the file.   
         .seek(<index>): to move the cursor to specified index of the opened file.  
         .readline(): to read one line of the opened file.  
         .readlines(): to get a list of all the lines of opened file.  
         .close(): to close the opened file.
      2. **Read, Write, Append**with open(<file\_name>) as <variable\_name>: file opened like this don’t need to be close.  
         mode=’r’: opening file in reading mode. This is default mode of open statement.  
         mode=’w’: opening file in writing mode. Returns the number of (characters written) indexes moved by cursor while writing. Clears the file before writing. Creates a new file if it doesn’t exist.  
         mode=:r+’: opening file in read + write mode. Writing with this mode, overwrites the existing content of the file without clearing it. It will throw an error if file doesn’t exist at the specified location.  
         mode=’a’: opening file in append mode. Returns the number of (characters written) indexes moved by cursor while writing. Writes the provided content at the end of existing file. Creates a new file if it doesn’t exist.
      3. **File Path**“.\\test\_folder\\test.txt”: to access file with relative file path (WINDOWS).  
         “./test\_folder/test.txt”: to access file with relative file path (LINUX).  
         “D:\…\\test\_folder\\test.txt”: to access file with absolute file path (WINDOWS).  
         “Users/…/test\_folder/test.txt”: to access file with absolute file path (LINUX).  
         Python module pathlib: [*https://docs.python.org/3/library/pathlib.html*](https://docs.python.org/3/library/pathlib.html)
      4. **File IO Errors**  
         FileNotFoundError: When file is not found at the specified path.  
         IOError: When system has difficulty reading or writing the file.
      5. **Exercise Translator:**Build an offline translator using Python modules and files.  
         Python Pypi translator module: [*https://pypi.org/project/translate/*](https://pypi.org/project/translate/)

**XXIII  
Bonus Extra Bits**

1. **Quick Note Upcoming Videos**[***The Complete Web Developer: Zero to Mastery***](https://www.udemy.com/course/the-complete-web-developer-zero-to-mastery/?referralCode=FFF295AECF3594CE440E)
2. **CWD Git + Github**GitHub is used for source control. It can help to counter merge conflict problem when more than one developer is working on same project.  
     
   Git official website: [*https://git-scm.com/downloads*](https://git-scm.com/downloads)# Git Bash:   
   git config –-global user.name “<Username>” #Set name  
   git config –-global user.email “<Email>” #Set email  
   git clone <repository path> #clone repository to local   
   git status #get status of local files  
   git add <file name> #add files to commit list  
   git add . #add all files to commit list  
   git commit –m“<message>” #commit files to GitHub  
   git push #finish commit to GitHub  
   git pull #grab latest commit from GitHub  
   git log #to get detailed commit history  
   git log –-oneline #to get brief commit history  
   git log graph #to get the graph of branches

**Bibliography**

[*https://zerotomastery.io/community/coding-challenges/*](https://zerotomastery.io/community/coding-challenges/)

[*https://zerotomastery.io/blog/*](https://zerotomastery.io/blog/)

[*https://zerotomastery.io/resources/*](https://zerotomastery.io/resources/)

[*https://zerotomastery.io/blog/?tag=PM&utm\_source=udemy&utm\_medium=coursecontent#monthly*](https://zerotomastery.io/blog/?tag=PM&utm_source=udemy&utm_medium=coursecontent#monthly)

[*https://zerotomastery.io/blog/?utm\_source=udemy&utm\_medium=coursecontent#monthly*](https://zerotomastery.io/blog/?utm_source=udemy&utm_medium=coursecontent#monthly)

[*https://www.linkedin.com/groups/12121940/*](https://www.linkedin.com/groups/12121940/)

[*https://www.linkedin.com/school/64685953/*](https://www.linkedin.com/school/64685953/)