**THIS DOCUMENT IS WRITTEN BY NYI NYI AUNG,**

**PRINCIPAL AT MANDALAY SOFTWARE INSTITUTE**

**https://www.facebook.com/mdysoftwareinstitute**

# Python Short Course

## Installation

1. https://www.python.org/download
2. https://code.visualstudio.com/download

## Python Extension Installation on VS Code


# Alternate Ways of Writing and Running Python Code

## Jupyter Notebook Installation

### Using Anaconda

https://www.anaconda.com/products/distribution

**For Experienced Python Users: Installing Jupyter with pip**

1. pip3 install –upgrade pip
2. pip3 install jupyter

**IDEs for Python**

https://colab.research.google.com

https://www.jetbrains.com/pycharm/

https://www.eclipse.org/downloads/

## VARIABLES

Variables are used to store information to be referenced and manipulated in a computer program. They also provide a way of labeling data with a descriptive name, so our programs can be understood more clearly by the reader and ourselves. It is helpful to think of variables as containers that hold information. Their sole purpose is to label and store data in memory. This data can then be used throughout your program.

## DATA TYPES IN PYTHON

1. Integer
2. Float
3. String
4. Boolean

## SEQUENCE DATA TYPES IN PYTHON

1. List
2. Tuple
3. Set
4. Dictionary

## NAMING VARIABLES

1. A variable name must start with a letter or the underscore character.
2. A variable name cannot start with a number.
3. A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9 and _ )
4. Variable names are case-sensitive (age, Age and AGE are three different variables)

```
#Legal variable names:
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"

#Illegal variable names:
2myvar = "John"
my-var = "John"
my var = "John"
```

**ASSIGNING VALUES TO VARIABLES**

first_number = 10

second_number = 2.5

first_name = “John”

last_name = ‘Smith’

operator = True

**OUR FIRST PROGRAM**

```
print("Hello World!")
```

**ADD TWO NUMBERS**

```
first_number = 15
second_number = 30

result = first_number + second_number

print("The result is", result)
```

**ARITHMETIC OPERATORS**

1. Addition ( + )
2. Subtraction ( - )
3. Multiplication ( * )
4. Division ( / ) → 5 / 2 = 2.5
5. Modulus ( % ) → 5 % 2 = 1
6. Floor Division ( // ) → 5 // 2 = 2
7. Exponent ( ** ) → 2 ** 3 = 8

## ASSIGNMENT OPERATORS

1. Assign ( = )
2. Add and assign ( += )
3. Subtract and assign ( -= )
4. Multiply and assign ( *= )

## COMPARISON OPERATORS

1. Equal to ( == )
2. Not equal to ( != )
3. Less than ( < )
4. Less than or equal ( <= )
5. Greater than ( > )
6. Greater than or equal ( >= )

## COMMENT

- Comments can be used to explain Python code.
- Comments can be used to make the code more readable.
- Comments can be used to prevent execution when testing code.

Comments starts with a #, and Python will ignore them:

```
#This is a comment
print("Hello, World!")

print("Hello, World!") #This is a comment

#print("Hello, World!")
print("Cheers, Mate!")
```

## MULTI LINE COMMENT

```
#This is a comment
#written in
#more than just one line
print("Hello, World!")

"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

## USER INPUT

**Example 1:**

```
first_name = input("Enter your first name: ")
family_name = input("Enter your family name: ")

full_name = first_name + " " + family_name

print("Your name is", full_name)
```

**Example 2:**

```
first_number = int(input("Enter the first number: "))
second_number = int(input("Enter the second number: "))

result = first_number + second_number

print("The result is", result)
```

**Example 3:**

```
first_number = input("Enter the first number: ")
second_number = input("Enter the second number: ")

result = float(first_number) + float(second_number)

print("The result is", result)
```

## STRING SLICING

```
my_string = "We are teaching Python"

print(my_string)
print(my_string[0])
print(my_string[0:5])
print(my_string[0:10:2])
print(my_string[-1])
print(my_string[-3:-1])
print(my_string.lower())
print(my_string.count("a"))
print(my_string.find("a"))
print(len(my_string))
```

## STRING FORMATTING

```
print("This is a string")

x = 3
print(x)

print(f"The current value of X is {x}")

my_string = "{first_name} Shakeshpere is a {job}."

print(my_string.format(first_name = "William", job = "poet"))
```

### Mini Project 1: Letter Counter

1. Print a welcome message to the user
2. Take a string (message) and a letter from the user
3. Count how many times this letter occurred
4. Calculate the percentage of the letter in the message
5. Print the outputs to the user

## LIST

```
my_list = [“apple”, “banana”, “cherry”]
print(my_list)
print(len(my_list))
print(type(my_list))
print(my_list[1])
print(my_list[-1])
print(my_list[2:5])
print(my_list[:4])
print(my_list[-4:-1])

if “apple” in my_list:
       print(“Ye, ‘apple’ is in the fruit list”)
```

```
my_list[1] = “orange”
print(my_list)


my_list = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
my_list[1:3] = ["blueberry", "watermelon"]
print(my_list)


my_list = ["apple", "banana", "cherry"]
my_list[1:2] = ["blackcurrant", "watermelon"]
print(my_list)


my_list = ["apple", "banana", "cherry"]
my_list[1:3] = ["watermelon"]
print(my_list)


my_list = ["apple", "banana", "cherry"]
my_list.insert(2, "watermelon")
print(my_list)


my_list = ["apple", "banana", "cherry"]
my_list.append("orange")
print(my_list)


my_list = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]
my_list.extend(tropical)
print(my_list)


my_list = ["apple", "banana", "cherry"]
my_list.remove("banana")
print(my_list)
```

```
my_list = ["apple", "banana", "cherry"]
my_list.pop(1) # Also check with my_list.pop()
print(my_list)


my_list = ["apple", "banana", "cherry"]
del my_list [0] # Also check with del my_list
print(my_list)


my_list = ["apple", "banana", "cherry"]
my_list.clear() # clear() method empties the list but the list still remains
print(my_list)


my_list = ["orange", "mango", "kiwi", "pineapple", "banana"]
my_list.sort()
print(my_list)


my_list = ["orange", "mango", "kiwi", "pineapple", "banana"]
my_list.sort(reverse = True)
print(my_list)


my_list = ["banana", "Orange", "Kiwi", "cherry"]
my_list.sort(key = str.lower)
print(my_list)


my_list = ["banana", "Orange", "Kiwi", "cherry"]
my_list.reverse()
print(my_list)


thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
print(mylist)
```

```
thislist = ["apple", "banana", "cherry"]
mylist = list(thislist)
print(mylist)
```

```
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]

list3 = list1 + list2
print(list3)
```

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

list1.extend(list2)
print(list1)
```

**Using the list() constructor to make a list:**

```
my_list = (("apple", "banana", "cherry"))
print(my_list)
```

list1 = ["abc", 34, True, 40, "male"]

**TUPLE**

```
tuple1 = ("abc", 34, True, 40, "male")
print(tuple1)
```

**THE TUPLE() CONSTRUCTOR**

```
thistuple = tuple(("apple", "banana", "cherry")) # note the double round-brackets
print(thistuple)
```

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

**Convert the tuple into a list to be able to change it:**

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)

print(x)
```

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.append("orange")
thistuple = tuple(y)
```

```
thistuple = ("apple", "banana", "cherry")
y = ("orange",)
thistuple += y

print(thistuple)
```

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.remove("apple")
thistuple = tuple(y)
```

```
thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple) #this will raise an error because the tuple no longer exists
```

**UNPACKING A TUPLE**

```
fruits = ("apple", "banana", "cherry")

(green, yellow, red) = fruits

print(green)
print(yellow)
print(red)
```

```
fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")

(green, yellow, *red) = fruits

print(green)
```

```
print(yellow)
print(red)


fruits = ("apple", "mango", "papaya", "pineapple", "cherry")

(green, *tropic, red) = fruits

print(green)
print(tropic)
print(red)


tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)


fruits = ("apple", "banana", "cherry")
mytuple = fruits * 2

print(mytuple)
```

**SET**

```
set1 = {"abc", 34, True, 40, "male"}

print(set1)
```

**THE SET() CONSTRUCTOR**

```
thisset = set(("apple", "banana", "cherry"))
print(thisset)


thisset = {"apple", "banana", "cherry"}

thisset.add("orange")

print(thisset)


thisset = {"apple", "banana", "cherry"}
tropical = {"pineapple", "mango", "papaya"}

thisset.update(tropical)
```

```
print(thisset)


thisset = {"apple", "banana", "cherry"}
mylist = ["kiwi", "orange"]

thisset.update(mylist)

print(thisset)


thisset = {"apple", "banana", "cherry"}

thisset.remove("banana")

print(thisset)


thisset = {"apple", "banana", "cherry"}

thisset.discard("banana")

print(thisset)


thisset = {"apple", "banana", "cherry"}

x = thisset.pop()

print(x)

print(thisset)


thisset = {"apple", "banana", "cherry"}

thisset.clear()

print(thisset)


thisset = {"apple", "banana", "cherry"}

del thisset

print(thisset)


set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}
```

```
set3 = set1.union(set2)
print(set3)


set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set1.update(set2)
print(set1)


x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

x.intersection_update(y)

print(x)


x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

z = x.intersection(y)

print(z)


x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

x.symmetric_difference_update(y)

print(x)


x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

z = x.symmetric_difference(y)

print(z)
```

**DICTIONARY**

```
thisdict = {
  "brand": "Ford",
  "electric": False,
  "year": 1964,
  "colors": ["red", "white", "blue"]
}
```

```
print(thisdict[“brand”])
```

**Duplicate values will overwrite existing values:**

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964,
  "year": 2020
}
print(thisdict)
```

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict["model"]
```

```
x = thisdict.get(“model”)
```

```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.keys()

print(x) #before the change

car["color"] = "white"

print(x) #after the change
```

```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.values()

print(x) #before the change

car["year"] = 2020

print(x) #after the change
```

```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.values()

print(x) #before the change

car["color"] = "red"

print(x) #after the change


car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.items()

print(x) #before the change

car["year"] = 2020

print(x) #after the change


car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.items()

print(x) #before the change

car["color"] = "red"
print(x) #after the change


thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.update({"year": 2020})


thisdict = {
  "brand": "Ford",
```

```
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)


thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.popitem()
print(thisdict)


thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict["model"]
print(thisdict)


thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict
print(thisdict) #this will cause an error because "thisdict" no longer exists.
```

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.clear()
print(thisdict)


thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = thisdict.copy()
print(mydict)


thisdict = {
  "brand": "Ford",
  "model": "Mustang",
```

```
  "year": 1964
}
mydict = dict(thisdict)
print(mydict)
```

**NESTED DICTIONARY**

```
myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}
```

```
child1 = {
  "name" : "Emil",
  "year" : 2004
}
child2 = {
  "name" : "Tobias",
  "year" : 2007
}
child3 = {
  "name" : "Linus",
  "year" : 2011
}

myfamily = {
  "child1" : child1,
  "child2" : child2,
  "child3" : child3
}
```

**PYTHON COLLECTIONS (ARRAYS)**

There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered, unchangeable*, and unindexed. No duplicate members.
- **Dictionary** is a collection which is ordered** and changeable. No duplicate members.

**List Exercise:**

list_quiz = [100, 30, 2, 15, -4, 3, 40, -9, 10]

1. Get the last 4 elements. → Store the output in a variable called last_4.
2. Get the highest value in the list. → Store the output in a variable called max_val. (Use max() function)
3. Sort the values in the list in ascending order. → Store the output in a variable called sorted_list. (Use sorted() function)
4. Add the sorted elements from task 3 to the original list. → Store the output in a variable called added_list.

**Tuple Exercise:**

tuple_exer = (1, 6, 4, 10, 2, 100, 4, "hi", 104.2, 4)

Count how many "4" we have in the provided tuple and store the count in a variable called tuple_four_count.

**Dictionaries Exercise:**

Create an empty dictionary for the grocery items. Then add the items as follows:

1. Apple → 4
2. Banana → 6
3. Cucumber → 2
4. Pizza → 2
5. Chocolate → 3

**CONTROL FLOW**

1. **Conditional Statements – if**

```
price = int(input("Enter the price: "))
qty = int(input("Enter the quantity: "))
amt = price * qty
if amt > 100:
    print("10% discount is applicable.")
    discount = amt*10/100
    amt = amt-discount
print("Amount payable: ", amt)
```

### 2. Conditional Statements – if … else

```
price = int(input("Enter the price: "))
qty = int(input("Enter the quantity: "))
amt = price * qty
if amt > 1000:
    print("10% discount is applicable.")
    discount = amt*10/100
    amt = amt-discount
    print("Amount payable: ", amt)
else:
    print("Amount payable: ", amt)
```

### 3. Conditional Statements – if … elif … elif … else

```
marks = int(input("Enter your marks: "))

if marks > 0 and marks < 40:
    print("Failed!")
elif marks >= 40 and marks < 60:
    print("Passed!")
elif marks >= 60 and marks < 80:
    print("Credit!")
elif marks >= 80 and marks <= 100:
    print("Distinction")
else:
    print("Out of range!")
```

## For Loop

```
my_list = [10,20,30,40,50,60]

for i in my_list:
    print(i)
```

```
for i in range(10):
    print(i)
print("For Loop has finished!")
```

```
for i in range(1, 10, 2):
    print(i)
print("For Loop has finished!")
```

**While Loop**

```
num = 0
while num <= 5:
    print(num)
    num = num + 1
```

```
num = 0
while num <= 5:
    num = num + 1
    print(num)
```

**EXERCISES:**

Leap Year Program:

1. Accept the year from user input.
2. Divide that year by 4

   စားလို့မပြတ်ရင် is not leap year

   စားလို့ပြတ်ရင်

   Divide that year by 100

   စားလို့ မပြတ်ရင် is a leap year

   စားလို့ ပြတ်ရင်

   Divide that year by 400

   စားလို့ပြတ်ရင် is a leap year

   စားလို့ မပြတ်ရင် is not a leap year

### Break Statement

```
my_list = [1, 2, 3, "stop", 5, "stop", 4.1]

for i in my_list:
    print(i)
    if i == "stop":
        print("The loop will stop!")
        break
print("This is outside of the loop!")
```

Please rewrite with “While Loop”

### Continue Statement

```
my_list = [1, 2, 3, "stop", 5, "stop", 4.1]

for i in my_list:
    print(i)
    if i == "stop":
        continue
    print(i)
```

### Enumerate Function

```
my_list = [1, 2, 's', 3.1]

print(list(enumerate(my_list)))
```

```
my_list = [1, 2, 's', 3.1]

for i, c in enumerate(my_list):
    print(i)
    print(c)
```

### Enumerate Function Exercise

```
names = ["Aung Aung", "Mg Mg", "Hla Hla"]
salaries = [1000, 2000, 3000]
```

We want to link each person with his salary in a new list called people_salaries as follow:

"Person Name $1000"

So, we want each element in the new list to be a string that contains both the person's name along with his salary.

You should use: str(), append(), enumerate() and for loops.

### List Comprehension

```
my_list_normal = []
for i in range(20):
    my_list_normal.append(i)
print(my_list_normal)

my_list = [i for i in range(20)]
print(my_list)
```

### List Comprehension Exercise

```
student_dictionary = {
    "Steve Allen":30,
    "Michael Smith":80,
    "John Paul":40,
    "James Henry":60
}
```

We want to filter out the students who passed – above 50 – and get their first names in a list called passed_student_names.

You need to do everything in one line, so you need list comprehension. This exercise is a little bit advanced, so take your time.

Hint: Use split() to get the first names and use items() to retrieve the items in dictionary.

## FUNCTIONS

```
def addition(first_num, second_num):
    result = first_num + second_num
    return result

output = addition(20, 30)
print(output)
```

### Function Exercises

1. Implement a function called raise_to_power that takes two parameters(inputs), a number and a power that this number should be raised to. For example, if we call raise_to_power(10, 2), it should return 100. If we call it raise_to_power(3, 3), it should return 27.
2. `students_names = [ "Micheal Ford", "John Buch" , "Isra Raul", "Messi Ronaldo"]`
   `students_grades = [ 80 , 53 , 90 , 100]`
   Implement a function called link_names_grades that take two lists called student_names and student_grades. This function should link each student with his grade in a dictionary called names_grades, which is the output of the function.
3. `numbers = [10, 20, 17, 13]`
   `even_or_odd = False`
   Implement a function called filter_even_odd that should take a list of integers called numbers and another parameter(input) called even_or_odd to specify even or odd numbers. If even_or_odd is False, then we will choose the EVEN numbers. If even_or_odd is True, we will choose the ODD numbers. We store the output in a variable called filter_list.

## ERROR HANDLING

```
first_num = input("Enter the first number: ")
second_num = input("Enter the second number: ")

try:
    first_num = int(first_num)
    second_num = int(second_num)

    result = first_num + second_num

    print("The result is", result)
except:
    print("Please use numbers only!")
```

### Python Advanced Functions

- Lambda Functions
- Functions *args and **kwargs
- Iterators
- Generators and yield keyword
- Map Function
- Filter Function

## Object Oriented Programming

### Classes & Objects

- Class: The blueprint of the objects.
- Objects: The instance of the class.
- Attribute: The object characteristics.
- Method: The actions (Functions) that the object can do.

```
class Car:
    def __init__(self, color, year, model):
        self.color = color
        self.year = year
        self.model = model
```

### Class vs Instance Attributes

```
class Car:
    def __init__(self, color, year, model):
        self.color = color
        self.year = year
        self.model = model

    number_of_wheels = 4

car1 = Car("Yellow", 2010, "BMW")
print(car1.model)

car1.model = 2
print(car1.model)
```

### Getters and Setters

```
class Car:
    def __init__(self, color):
        self.__color = color

    def setColor(self, color):
        self.__color = color

    def getColor(self):
        return self.__color

car1 = Car("Yellow")
print(car1.getColor())

car1.setColor("Blue")
print(car1.getColor())
```

## INHERITANCE

```
class Vehicle():
    def __init__(self, color):
        self.__color = color

    def print_hello(self):
        print("Hello")

class Car(Vehicle):
    print("This is in Car class.")

car1 = Car("Yellow")
car1.print_hello()
```

### Method Overriding

```
class Vehicle():
    def __init__(self, color):
        self.__color = color

    def print_hello(self):
        print("Hello")

class Car(Vehicle):
    def print_hello(self):
        print("Not Hello!")
```

```
car1 = Car("Yellow")
car1.print_hello()
```

### Method Adding

```
class Vehicle():
    def __init__(self, color):
        self.__color = color

    def print_hello(self):
        print("Hello")

class Car(Vehicle):
    def print_hello(self):
        print("Not Hello!")

    def car_specific(self):
        print("I am a car.")

car1 = Car("Yellow")
car1.print_hello()
car1.car_specific()
```

### Super Function

```
class Vehicle():
    def __init__(self, color):
        self.__color = color

    def print_hello(self):
        print("Hello")

class Car(Vehicle):
    def print_hello(self):
        super().print_hello()
        print("Not Hello!")

    def car_specific(self):
        print("I am a car.")

car1 = Car("Yellow")
car1.print_hello()
car1.car_specific()
```

### Multi Inheritance

```
class Vehicle():
    def __init__(self, color):
        self.__color = color

    def print_hello(self):
        print("Hello")

class Factory():
    def __init__(self):
        pass

    def print_factory(self):
        print("This is in Factory class.")

class Car(Vehicle, Factory):
    def print_hello(self):
        super().print_hello()
        print("Not Hello!")

    def car_specific(self):
        print("I am a car.")

car1 = Car("Yellow")
car1.print_hello()
car1.car_specific()
car1.print_factory()
```

EXERCISES

1. Use “for loop” and expected result is:

   *

   * *

   * * *

   * * * *

   * * * * *

2. Print the output of the result of addition of from 1 to 100.
3. Accept the user input and check even or odd start from 1.

   If user type the number 5, your output look like this:

   1 is odd

   2 is even

   3 is odd

4 is even

5 is odd

4. Find the maximum number in the given list. Don't use max() function. Use "for loop".
   my_list = [1, 5, 2, 7, 8, 9, 200, 155]
5. Using the above list, accept the user input and you need to check the number that the user typed is in the list or not. And print out the result. If you find the number that the user typed in your list, you also need to print out the result with index number.
6. Implement triangle_star() function. Accept user input. If user typed the number 3, your output like this:
   *
   * *
   * * *
7. Create a class of MyCal class which include addition, subtraction, multiplication, and division of two number. Then create a child class called MyCalNext which include the calculation of square value.
8. Write a Python program which accepts the radius of a circle from the user and compute the area.

   from math import pi (Use that statement for pi)

9. Write a Python program to calculate the sum of three given numbers, if the values are equal then return three times of their sum.

10. Write a Python program to test whether a passed letter is a vowel or not.

11. Write a Python program to print all even numbers from a given numbers list in the same order and stop the printing if any numbers that come after 237 in the sequence.

```
numbers = [
            386, 462, 47, 418, 907, 344, 236, 375, 823,
            566, 597, 978, 328, 615, 953, 345,
            399, 162, 758, 219, 918, 237, 412, 566,
            826, 248, 866, 950, 626, 949, 687, 217,
            815, 67, 104, 58, 512, 24, 892, 894, 767,
            553, 81, 379, 843, 831, 445, 742, 717,
            958,743, 527
          ]
```