Smart way to Learn Python

by Mark Myers

Muhammad Hassan
Data science enthusiastic

What is Python

1. Python is high level computer programming language
2. It is general purpose programing language, Created by Guido van Rossum.
3. it's reasonably easy to learn, and it's relatively easy to read.
4. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small- and large-scale projects.
5. Its design philosophy emphasizes code readability with the use of significant indentation.
6. Python is popular, for 30 year old.

So it is enough now we direct move to practical side

What is “Print” in python

Print

?

Print

In Python, the command print tells the program to display words or numbers on the screen. Here's a line of code that tells Python to display the words “Hello, World!” Print is a keyword

```
print(“Hello, World!”)
```

Or

```
print(‘Hello, World!’)
```

One more thing

```
print(“Pakistan”, “Gilgit”, “KPK” , sep = “-“)  #what will be the output?
```

Now practic

Find the interactive coding exercises for this chapter at:

http://www.ASmarterWayToLearn.com/python/1.html

Variable for String

A variable is memory location. Also known as identifiers.

For example.

Name = “hassan”

but while creating variables - Name, name are two different variable.

name = “marks”

print(name)

What is the output?

```
name = “hassan”

name  = “ali”

print(name)
```

?

Now it is time to practice

Find the interactive coding exercises for this chapter at

http://www.ASmarterWayToLearn.com/python/2.html

Variable for Number

You can also assigned variable to a number like

weight = 130

Now you can make calculation to this variable “weight” like if you want to add 20 to this variable weight

weight + 20

initial_num = 50

new_num = intial_num + 20

Even you can make calculation even of variable like

sum = initial_num + new_num

General information regarding Variable

a = 10

So here

a is variable name (identifier)

= assignment operator

10 value (integer value)

Variable name legal or Illegal

You've already learned three rules about naming a variable:

1. You can't enclose it in quotation marks.

2. You can't have any spaces in it.

3. It can't be a number or begin with a number.

In addition, a variable can’t be Python reserved words or keywords.

Like print, False, return and so on..

What can’t be variable

1. Reserved word
2. Built in Library

Maths Expressions (Operators)

You can use different operators in python just like +(add), -(Substract), *(Multiplications) and /(Division).

The programming term for calculations is math expression.

For example.

num_1 = 4

Or num_1 = 2+2 both are same. You can also writ

print(2+2)

Continue..

num = 10

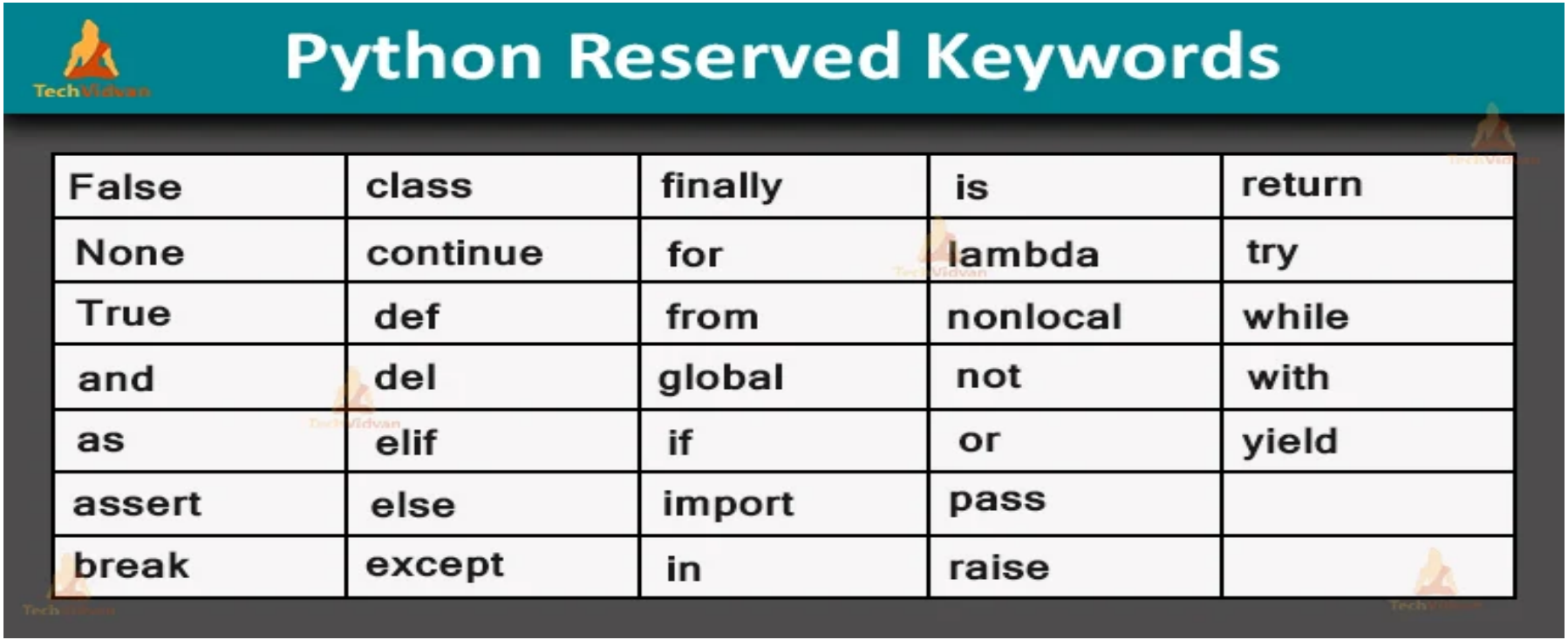
another_num = 1.5

sum_of_numbers = num + another_num

Now it is time to practice the knowledge..

http://www.asmarterwaytolearn.com/python/4.html

Keywords or Reserved words list.



One more thing to understand

Python's governing body recommends breaking up multi-word variables with underscores. That's what I'll ask you to do with your own variable names. It'll make them more readable, and you'll be less likely to get variables mixed up.

Examples:

user_response

user_response_time

user_response_time_limit

Now it is your turn

Find the interactive coding exercises for this chapter at

http://www.ASmarterWayToLearn.com/python/5.html

Math expressions: Unfamiliar operators

1. **modulo operator**

What is left over when one number is divided by another number.

For example

Left_over = 14 % 3 or ans = 99 % 4

But

If one number divides evenly into another, the modulo statement assigns 0 to the variable

left _over = 12 % 6 , the answer is will be assigned 0

Math expressions: Unfamiliar operators

In this lesson we will also discuss some other ways to write a code with mathematical operators.

2. Increments

Suppose you want to increase the value of a variable by 1. You could write...

age = age + 1

Another way is shorthand way

age += 1

This is same for division, multiplications, subtraction and additions

age *= 3 age -= 5

It is time to practice

Below link is the practice exercise.

http://www.ASmarterWayToLearn.com/python/6.html

Math expressions: Eliminating ambiguity

Complex arithmetic expressions can pose a problem, one that students face in high school algebra.

Look at this example and tell me what the value of total_cost is.

total_cost = 1 + 3 * 4

What is the value of total_cost ?

Some more examples

1. total_cost = 1 + (3 * 4)
2. total_cost = (1 + 3) * 4
3. result_of_computation = (2 * 4) * 4 + 2

But there is problem with last example (#3) and what is that?

Concatenating text strings

We use “+” sign to concatenate the string in python. When we want to concatenate two or more string

For example

greeting = “Hello”

place = “World”

print(greeting + place)

Practice here : http://www.ASmarterWayToLearn.com/python/8.html

if statements

Suppose you want to know whether the string assigned to the variable species is "cat."

This is the code.

```
if species == "cat":
    print("Yep, it's cat.")
```

It begins with the keyword if. Note that if is all lowercase.

Note that it's two equal signs, ==, not one. One equal sign, =, can only be used to assign a value to a variable

Now it’s time to practice

If statement - running a block of code, when some specific condition is satisfied.

Find the interactive coding exercises for this chapter at:

http://www.ASmarterWayToLearn.com/python/9.html

Comparison operators

comparison operator == and !=

== specifically it is equality operator.

Let see some examples

```
if full_name == first_name + " " + "Myers":
if full_name == first_name + " " + last_name:
if total_cost == 81.50 + 135:
if total_cost == materials_cost + 135:
```

When you're comparing strings, the equality operator is case-sensitive. "Rose" does not equal "rose." It's true that "Rose" != "rose".

Now it's time to practice

Find the interactive coding exercises for this chapter at

http://www.ASmarterWayToLearn.com/python/10.html

else and elif statements

The if statements you've coded so far have been all-or-nothing. If the condition tested true, something happened. If the condition tested false, nothing happened.

```
if species == "cat":
    print("Yep, it's cat.")
if species != "cat":
    print("Nope, not cat.")
```

Still there is problem and what is the problem with above code?

```
if species == "cat":
    print("Yep, it's cat.")
else:
    print("Nope, not cat.")
```

Elif is short form of else if

Finally, there's elif. It's short for else if. If no test has been successful yet, an elif tries something else.

```
 if donut_condition == "fresh":
        buy_score = 10
 elif donut_price == "low":
       buy_score = 5.5
else:
        buy_score = 0
```

Now it's time to practice

Find the interactive coding exercises for this chapter at

http://www.ASmarterWayToLearn.com/python/11.html

Testing sets of conditions

Using the if statement, you've learned to test for a condition. If the condition is met, one or more statements execute. But suppose not one but two conditions have to be met in order for a test to succeed.

For example:

```
if weight > 300 and time < 6:
        status = "try to recruit him"
```

Simple example:

```
if  marks >=70 and marks <=80:
                print(“Secured A grade”)
```

You have learn AND OR operator here.

```
if SAT > avg or GPA > 2.5 or parent == "alum":
        message = "Welcome to Leeds College!"
        print(message)


if (age > 65 or age < 21) and res == "U.K.":
```

Practice here:

http://www.ASmarterWayToLearn.com/python/12.html

if statements nested

List

List is same as variable but list has the modification way and some additional features of list.

For example.

cities = [“karachi”, “lahore”, “queeta”, “Peshawar”]

In the list we have different data types as well like.

list_ele = [“karachi”, 1, “abc” , 8.0]

In the above examples “cities” and list_ele are list name and under square brace are elements of list.

While defining a list Things to keep in mind.

1. The first element in a list always has an index of 0, not 1. This means that if the last element in the list has an index of 9, there are 10 items in the list.
2. The same naming rules you learned for ordinary variables apply. Only letters, numbers, and underscores are legal. The first character can't be a number. No spaces.
3. It's a good idea to make list names plural—cities instead of city, for example—since a list usually contains multiple things.

Lists: Adding and changing elements

append function is use to add new element in list, just like.

cities = [“karachi”, “lahore”, “queeta”, “Peshawar”]

So if we want to add new city “Islamabad”

cities.append(“Islamabad”)

There's an alternative way to append. It allows you to add one or more elements to a list.

cities = cities + [“Multan”, “Gilgit”, “Faisalabad”]

If i want to add element in any index like i want add “Sialkot” in the beginning than we used “insert” function.

cities.insert(0, “Sialkot”)

Lists: Taking slices out of them

If you need range of elements from list you will use slicing just like,

cities = [“Karachi”, “Lahore”, “Queeta”, “Peshawar”, “Gilgit”, “Multan”]

You can copy elements 2 through 4 to create another list... smaller_list_of_cities = cities[2:5] output will be Queeta, Peshawar, Gilgit.

When you slice from a list, the list is unchanged. Think "copy," not "cut."

When the first element of the slice is the first element of the original list— the element with an index of 0—you can omit the first number altogether: smaller_list_of_cities = cities[:5]

When the last element of the slice is the last element of the original list, you can omit the second number: smaller_list_of_cities = cities[2:]

Lists: Deleting and removing elements

Let suppose you a list name cities..

cities = [“karachi”, “Lahore”, “Quetta”, “Peshawar”]

Now if you want to delete Lahore, you will use

del cities[1]

After delete the element in index 1, Python adjusts the index numbers so there are no gaps. The new list will be

cities = [“karachi”, “Quetta”, “Peshawar”]

The statement begins with the keyword del, short for delete:

Lists: Deleting and removing elements

Now you can also strike an element off a list by specifying its value instead of its index number:

cities = [“karachi”, “Lahore”, “Quetta”, “Peshawar”]

cities.remove(“Lahore”)

The statement begins with the keyword remove after remove parenthesis and then the string or number to remove.

Lists: popping elements

On the other hand if you want to pop the element and add to another list of variable you will use the key word “pop”.

For example.

cities = [“karachi”, “Lahore”, “Quetta”, “Peshawar”]

cities.pop(2)

Quetta will pop from the list.

Tuples

Tuple is data type like list, but the elements are fixed. They can't be changed—unless you redefine the whole tuple.

For example, if you want to fix the collection of province of Pakistan

pro_pak = (“Punjab”, “Sindh”, “KPK”, “Balochistan”)

Tuple use parentheses to define the data type.

To pick the second element in the tuple you can..

Pro_pak[1],

Sindh will be the answer.

Tuples

You can't add, modify, remove, delete, or pop. If you must make a change, you have to define the tuple all over again.

If the order of elements changes for any reason, you have to re-code the whole tuple, too.

for loops

Just take an example

cities_pak = [“karachi” , “Islamabad”, “quetta”, “peshawar”, “lahore”, “multan”,”hyderabad”]

city_to_check_ = “peshawar”

```
for city_to_check in cities_pak:
        if city_to_check == cities_pak
            print(“it is city of pakistan”)
```

In simple English what does loop tell us:

1. for each element, one at a time, in the list:
2. do something with that element

for loops

In for loop if the required result found you can stop the loop for further iteration by the key word. “break”.

```
cities_pak = ["karachi" , "Islamabad", "quetta", "peshawar", "lahore",
"multan","hyderabad"]
city_to_check  =  "peshawar"
for a_city_to_check in cities_pak:
  if city_to_check == a_city_to_check:
    print("it is city of pakistan")
    break
  else:
    print("It is not city of pakistan")
```

for loops nested

Take an example to understand it.

```
first_names = ["Mohd", "Zahid", "Asad", "Shabbir", "Tanveer"]
last_names = ["Hassan", "Ali", "Hussain", "wali", "Khan"]
full_name = []

for a_first_names in first_names:
  for a_second_names in last_names:
    full_name.append(a_first_names + " "+ a_second_names)
print(full_name)
```

Getting information from the user and converting strings and numbers

Getting information from the user, any thing string , number or float you can get from the user using “input” keyword.

for example;

user_name = input(“Enter your name: ”)

Enter your name:

Changing case

```
user_input = input(“Enter you country name:”)

my_name = “muhamamd hassan”

my_name.lower()          #it will convert all the letter in the string to lowercase

my_name.upper() # it will convert all the letter into uppercase

my_name.title() #title will convert all first character in the sentence into upper case
```

Dictionaries

A dictionary works something like a list, but instead of a simple series of things, a dictionary is a series of pairs of things. Each pair contains a key.

Suppose we have customer data:

custome_id = 0101

customer_first_name = “Hasnan”

customer_second_name = “Alam”

customer_address = “pakistan”

Dictionaries

Code will be looks like:

```
customer  = {
                        “customer_id”  : 0101,
                        “customer_first_name “ : “Hasan”,
                        customer_second_name  : “Alam”,
                        customer_address  :  “Pakistan”
}
```

The series is enclosed in brackets—but in a list, brackets are square brackets, and in a dictionary, brackets are curly brackets.

Dictionary hold pair of key and values.

Dictionaries: How to pick information out of them

In the list we are picking the element by its index, but dictionary has no any index concept so here we will pick the element by its key. Like

```
customer  = {
                        “customer_id”  : 0101,
                        “customer_first_name “ : “Hasan”,
                        customer_second_name  : “Alam”,
                        customer_address  :  “Pakistan”
}
address_of_customer = customer[“cutomer_address”]
print(address_of_customer)
```

Dictionaries: The versatility of keys and values

Keys can be string and number integers. Like

ranking = {1 : “USA”, 2 : “UK”, 3 : “Germany”, 4 : “China” }

What if you need ranking third country

third_ranking_country = ranking[3] you will received “UK”.

You can also mix the values number than string.

Dictionaries: Adding items

Let take previous example:

```
customer  = {
                        “customer_id”  : 0101,
                        “customer_first_name “ : “Hasan”,
                        customer_second_name  : “Alam”,
                        customer_address  :  “Pakistan”
}
customer[“gender”] =  “male”
```

You can add gender to the dictionary customer.

Dictionaries: Looping through values

If you want to display the keys in dictionary you will replace the values with keys, like:

```
for each_key in customers.keys():

        print(each_key)
```

Single difference the word keys and values.

Dictionaries: Looping through key value pairs

You have learnt loop through keys and values but here you will learn loop through both keys and values in a single line of code, with the help of items keyword. like

```
for        each_keys, each_values in customers.items():

           print(“The customer’s “ +each_keys+ “is” +each_value)
```

Creating a list of dictionaries

List of dictionaries

Take an example of customer

```
customer  =[
        {
                “Customer_id” : 0,
                “customer_first_name”: “Mohd”,
                “customer_second_name”: “Hassan”,
}
]
```

How to pick information out of a list of dictionaries

A list that contain dictionary are called list of dictionary.

Like

```
customers = [
        { "customer id": 0,
        "first name":"John",
        "last name": "Ogden", },
        { "customer id": 1,
        "first name":"Ann",
         "last name": "Sattermyer", }]
```

How to pick information out of a list of dictionaries

So here as you know list has index value as python assigned automatically.

Above example customer 0 will be assigned index 0 and customer 1 will be 1 and so on..

For example

dic_to_look_in = customer[1]

One constraint presented by this scheme: If you lose a customer, you can't delete her dictionary from the list. If you do, index numbers of dictionaries will change. They'll stop matching the customer ids inside them.

How to append a new dictionary to a list of dictionaries

```
new_dictionary =
        { "customer id": new_customer_id,
         "first name": new_first_name,
         "last name": new_last_name,
         "address": new_address, }
```

Finally, we append this new dictionary to the list:

```
customers.append(new_dictionary)
```

Creating a dictionary that contains lists

Suppose we have a dictionary of customer.

```
customer_29876 = {
                                    "first name": "Mohd",
                "last name": "Hassan",
                                    "address": "Gilgit, Pakistan", }
```

Let say we offered this customer different discounts, and he qualified three different discount, so we add these discounts in list.

```
customer_29876 = {
                                    "first name": "Mohd",
                "last name": "Hassan",
                                    "address": "Gilgit, Pakistan",
                “Discounts” : [“stanard”, “volumn”, “loyalty”] }
```

How to get information out of a list within a dictionary

If we are giving the following discount to the customer.

brother-in-law – 30%

loyalty – 15%

volume – 10%

standard – 5%

```
if "brother_in_law" in customer_29876["discounts"]:
  discount_amount = 0.3
elif "loyalty" in customer_29876["discounts"]:
  discount_amount = .15
elif "volumn" in customer_29876["discounts"]:
  discount_amount = .10
elif "standard" in customer_29876["discounts"]:
  discount_amount = .5
```

Creating a dictionary that contains a dictionary

```
customers = {                                                    #we just
eliminate the customer_id
 0: {
 "first name":"John",
 "last name": "Ogden",
 "address": "301 Arbor Rd.",
 },
 1: {
 "first name":"Ann",
 "last name": "Sattermyer",
 "address": "PO Box 1145",
 },
 2: {
 "first name":"Jill",
 "last name": "Somers",
 "address": "3 Main St.",
 },
 }
```

Creating a dictionary that contains a dictionary

It is not necessary to put key a number we can change like.

```
customers = {
 "johnog": {
 "first name":"John",
 "last name": "Ogden",
 "address": "301 Arbor Rd.",
 },
 "coder1200": {
 "first name":"Ann",
 "last name": "Sattermyer",
 "address": "PO Box 1145",
 },
 "madmaxine": {
 "first name":"Jill",
 "last name": "Somers",
 "address": "3 Main St.",
 },
```

How to get information out of a dictionary within another dictionary

```
customers = {
 "johnog": {
 "first name":"John",
 "last name": "Ogden",
 "address": "301 Arbor Rd.",
 },
 "coder1200": {
 "first name":"Ann",
 "last name": "Sattermyer",
 "address": "PO Box 1145",
 },
 "madmaxine": {
 "first name":"Jill",
 "last name": "Somers",
 "address": "3 Main St.",
 },
 }
print(customers[“madaxine”][“address”]
```

Functions

A function is a block of Python code that robotically does the same thing again and again, whenever you invoke its name. It saves you repetitive coding and makes your code easier to understand.

Function declaration or definition

```
def add_number():
  num_1 = 7
  num_2 =11
  total = num_1 + num_2
  print(total)
```

Function calling

```
add_number()
```

In general

Parameters & Arguments

Parameters are expected values given from user

Arguments are values that user supplies to function

Functions: Passing them information

If you put some information inside the parentheses, that information is passed to the function. The function can then use the information when it executes. Like **def num(num_1, num_2):**

definition

```
def add_number(number_1, number_2):
  total = number_1 + number_2
  print(total)
```

Function calling

```
add_number(33, 55)
```

Arguments that are loaded into function parameters in order are Positional Argument.

Functions: Passing information to them a different way

If you don't have to match arguments to parameters by order. You can code **keyword arguments.**

In this form of function each argument begin with key, the function definition load that value into correct parameter, not according to order, but by matching key in the function.

Example:

```
say_names_of_couples(husband= "Hassan", wife="unkown")


def say_names_of_couples(husband, wife):
  print("The name of the couple are "+ husband+ "" + wife)
```

Functions: Assigning a default value to a parameter

Note:

Only keyword parameters can have a default value. Positional parameters can't.

Variable assigning

```
sales_total = 101.35
tax_rate = 0.05
```

Definition of function

```
def cal_tex(sales_total, tax_rate):
  print(sales_total *  tax_rate)
```

Calling Function

```
cal_tex(sales_total, tax_rate)
```

Functions: Mixing positional and keyword arguments

You can mix positional arguments and keyword arguments. For example, you can code...

give_greeting("Hello there", first_name="Al")

```
def give_greeting(greeting, first_name):
    print(greeting + ", " + first_name)
```

Be careful mixing positional and keyword arguments.

Positional arguments must come before keyword arguments.

Functions: Dealing with an unknown number of arguments

You have learnt function with positional argument and keyword arguments But suppose there are times when you want to display other information about the function—but not always. For example, you call the same function and include others optional arguments:

```
def display_match(winer, score, **other_info):
  print("The winer is ", winer)
  print("The score is = ", score)
  for key, value in other_info.items():
    print(key + ": " +value)

display_match(winer="Real Madrid", score="2-1", overtime="yes", injures="none", weather="normal")
```

The two asterisks followed by a parameter name mean that there may or may not be one or more extra arguments passed.

Note: Optional arguments must come after regular arguments. Optional parameters must come after regular parameters. Positional arguments can be optional as well.

Optional Keyword argument are with ** while Optional parameter argument are with *

Functions: Passing information back from them

In order to pass information back to the calling code, you need two things. You need a final line in the function that sends the information back to the calling code and you need a way for the calling code to accept the information. In this case, it's a variable:

```
def cal_tex(sales_total, tax_rate):
  return(sale_total * tax_rate)

sales_tax = calc_tax(sales_total=101.37, tax_rate=.05)
print(sales_tax)
```

The last line can be written into one line like

```
print(calc_tax(sales_total=101.37, tax_rate=.05))
```

Using functions as variables (which is what they really are)

We can write the result of function as

```
def add_numbers(first_number, second_number):
  return first_number + second_number
def substract_numbers(first_number, second_number):
  return first_number - second_number
#Calling functions
sum_of_results = add_numbers(1, 2) + subtract_numbers(3, 2)
```

Functions: Local vs. global variables

Now its time to discuss the scope of variable and these scope are **local** and **global.**

A global variable is one you define in the main body of your code—that is, not in a function.

```
what_to_say = "Hi"
```

A local variable is one that you define in a function.

```
def say_something():

  what_to_say = "Hi"
```

Local Variables: All the variables defined inside a function body are local and CAN Not be accessed outside the function

Global Variables: All the variables defined outside a function body are global and CAN be accessed outside as well as inside the function body

Functions within functions

Within a function you can call another function.

Note: The function that is called must be earlier in your code than the function that calls it.

Advantages of Functions.

code reusability

code simplicity/readability

bug resolving

While loops

Suppose you want you print or display digits from 1 to 10.

```
a = 0
while a<=10:
    print(a)
    a +=1
```

While loop start with the initial value than a condition .

While loops: Setting a flag

Let take an example.

```
myguests = []

flag = True

while flag:

    guestName = input("Enter a guest name or press 'q' to exit")

    if guestName=="q":

        flag=False

    else:

        myguests.append(guestName)

Myguests
```

True and False aren't enclosed in quotation marks.

They aren't strings. They must be capitalized.

Classes

In python classes are **templates**, they help you **standardized** and **organize** information.

Things to notice about this first line of code that defines a class: It begins with the keyword **class.**

```
class  Patient():
```

Then comes the name you're giving it. Naming rules here follow those for variables. The only difference is that by custom, a class name begins with a capital letter:

The class name is followed by a pair of parentheses and a colon:

Classes: Starting to build the structure

Class structure will be like

```
class Patient():
  def __init__(self, last_name):
```

Note that line 2 is indented

Classes: A bit of housekeeping

Let see the code.

```
class Patient():
  def __init__(self, last_name):
    self.last_name = last_name
```

We have add the last line. The last name is followed by self and dot.

Classes: Creating an instance

```
class Patient():
  def __init__(self, last_name):
    self.last_name = last_name

pid10101 = Patient('Hassan')
```

The Patient class is created above with one parameter called last_name and while calling we have created instance now it is unique for all patient.

Just like.

```
pid10102 = Patient('Talib')
```

Classes: A little more complexity

Let’s we have add some more parameters to the class and functions like

```
class Patient():
  def __init__(self, last_name, first_name, age):
    self.last_name = last_name
    self.last_name = first_name
    self.age = age

pid10101 = Patient('Hassan', "ali", 77)
pid10102 = Patient('Talib', 'hussain', 66)
Its is like positional argument call the value according to position of parameters.
```

Classes: Getting info out of instances

```
class Patient():
  def __init__(self, last_name, first_name, age):
    self.last_name = last_name
    self.last_name = first_name
    self.age = age

pid10101 = Patient('Hassan', "ali", 77)
pid10102 = Patient('Talib', 'hussain', 66)

If we need the age of patient Talib we can call
Age_of_patient = pid10102.age
Or
print(pid10102.age)
```

Classes: Building functions into them

Method: When we can build the function into the class itself, function is called a method.

```
class Patient():
  def __init__(self, last_name, first_name, age):
    self.last_name = last_name
    self.last_name = first_name
    self.age = age

pid10101 = Patient('Hassan', "ali", 77)
pid10102 = Patient('Talib', 'hussain', 66)

Just like we have define the function

def say_if_minor(patient_first_name, patient_last_name, patient_age):
  if patient_age < 21:
    print(patient_first_name + " " + patient_last_name + " is a minor")

This function is call method.
```

Classes: Coding a method

```
class Patient():
  def __init__(self, last_name, first_name, age):
    self.last_name = last_name
    self.first_name = first_name
    self.age = age
  def say_if_minor(self):
    if self.age < 21:
      print("This patient is a minor")
```

Classes: Changing an attribute's value

If you want to change the record of patient like the last name change to "Akram", from "Hassan". so this was the record:

```
pid10101 = Patient('Hassan', "ali", 77)

Pid10101.last_name = "Akram". #this work right but we will do it from function
```

```
class Patient():
  def __init__(self, last_name, first_name, age):
    self.last_name = last_name
    self.first_name = first_name
    self.age = age
  def say_if_minor(self):
    if self.age < 21:
      print("This patient is a minor")
  def change_last_name(self, new_last_name):
    self.last_name = new_last_name
```

Reviewer links

Kaggle account: https://www.kaggle.com/dshassan

Linkedin id : https://www.linkedin.com/in/muhammad-hassan-4839811b7/