python-basics-assignment

January 8, 2023

Python Basics

[1]:
```
# What is your name! print your name!
# Only use one print function

print("Akshay Anand")
```

```
Akshay Anand
```

- () <= Parentheses
- '' <= Single Quotes
- "" <= Double Quotes
- \n <= New_line
- # <= Used to comment inside code

[2]:
```
# define variables named as with values: mukesh=7, z=6, rohan=5, longitude=4
mukesh=7
z=6
rohan=5
longitude=4
```

[8]:
```
# print required variable
# output - 5
print("rohan",rohan)
```

```
rohan 5
```

Variable Assignment: **`Variable_Name = Value`**

Variables Naming Rules: * `Python is case-senstive` => x=5 is different from X=5 (one is lowe and other is upper case) * `var name can't start with special character except underscore(_)` => _X = 7 is `valid`, @X = 7 is `invalid` * `var name can't start with number` => 9X = 7 is `invalid`, X9 = 7 is `valid` * can't use `keywords` as a variable name *

1 Declaring a Variable

[9]:
```
# declare 4 variables with values as: ur_age 21,ur_weight 50.6, ur_first_name =␣
↪'Mukesh',ur_last_name = "Manral"
ur_age=21
```

```
ur_weight=50.6
ur_first_name='Mukesh'
ur_last_name="Manral"
```

2 Data Type(Type of variable)

Name	Type	Description
Integers	int	Integer number, like 34,-56 …
Float	float	Decimal number, like 3.4,-5.6 …
String	str	Ordered sequence of characters, like 'your name'
Boolean	bool	Logical values indicating True or False only

[13]:
```
# print type of ur_age,ur_weight,ur_first_name,ur_last_name variables
type(ur_age)
```

[13]:
```
int
```

[14]:
```
type(ur_weight)
```

[14]:
```
float
```

[15]:
```
type(ur_first_name)
```

[15]:
```
str
```

[16]:
```
type(ur_last_name)
```

[16]:
```
str
```

[17]:
```
# print values of ur_age,ur_weight,ur_first_name,ur_last_name variables
ur_age,ur_weight,ur_first_name,ur_last_name
```

[17]:
```
(21, 50.6, 'Mukesh', 'Manral')
```

[18]:
```
# make 2 variables with values as: ur_first_name 'Mukesh',ur_last_name'Mukesh'
ur_first_name='Mukesh'
ur_last_name='Mukesh'
# make a variable TrueOrFalse which will have comparison of variables␣
 ↪ur_last_name == ur_first_name
TrueOrFalse=(ur_last_name==ur_first_name)
```

[19]:
```
TrueOrFalse
```

[19]:
```
True
```

[20]:
```
# define a variable name "x" and assign value 777 and print it
x=777
print(x)
```

777

- To view some data on screen, python have `print` function
  - Using `print` function we can control view on output screen

[ ]:
```

```

`Operators`: Symbols that represent mathematical or logical tasks

Example: 700 + 77 * + <= Operator * 700 & 77 <= Operands

[21]:
```
# Initialize variables [x,y,z,zz] with values
## x as 7 =>int ,
## y as 77 =>int,
## z as 77.7 => float,
## zz as 'Hi' => string
x=7
y=77
z=77.7
zz='Hi'
```

3 Arithmetic Operators

[22]:
```
# add x and z
x+z
```

[22]: 84.7

[23]:
```
# subtract z and y
z-y
```

[23]: 0.7000000000000028

[24]:
```
# Multiply x and z
x*z
```

[24]: 543.9

[25]:
```
# Exponent (raise the power or times) x times z
x**z
```

[25]: 4.614426248242042e+65

[26]:
```
# division on x and z
x/z
```

[26]:
```
0.09009009009009009
```

// => divides and returns integer value of quotient * It will dump digits after decimal

[27]:
```
# floor division(ignores decimal) on x and z (gives quotient)
x//z
```

[27]:
```
0.0
```

[28]:
```
# Modulo(gives remainder) on x and z
x%z
```

[28]:
```
7.0
```

4 Comparison Operators

[29]:
```
# comapre and see if x is less then z
# can use '<' symbol

x<z
```

[29]:
```
True
```

[30]:
```
# check the type of above comaprison where it says comapre and see if x is less␣
↪then z
type(x<z)
```

[30]:
```
bool
```

- `Bool` => takes two values, either `True` or `False`

[31]:
```
# compare and see if x is less then or equall to z
# can use '<=' symbol
x<=z
```

[31]:
```
True
```

[32]:
```
# comapre and see if x equall to z
# can use '==' symbol
x==z
```

[32]:
```
False
```

[33]:
```
# comapre and see if x is greater than z
# can use '>' symbol
x>z
```

[33]: False

[34]:
```
# comapre and see if x is greater than or equall to z
# can use '>=' symbol
x>=z
```

[34]: False

[35]:
```
# comapre and see if x is Not equall to z
# can use '!=' symbol
x != z
```

[35]: True

5 Logical Operators

[36]:
```
# compare if 108 is equall to 108, 21 is equall to 21 using logical and
# equall to => '=='
# logical and => and

# in and both condition must be True to get a True
108==108 and 21==21
```

[36]: True

[37]:
```
# how above condition can give False as output show all those conditions
108 != 108 and 21==21
```

[37]: False

[38]:
```
108==108 and 21 != 21
```

[38]: False

[39]:
```
108 != 108 and 21 != 21
```

[39]: False

[40]:
```
# compare if 108 is equall to 108, 21 is equall to 11 using logical or
# equall to => '=='
# logical or => or
```

```
# in or Only one condition need to be True to get a True
108==108 or 21==21
```

[40]: True

[ ]:
```
# this is for you to understand it
(108 == 108) or (21 == 11) or (108 <= 11)
```

[ ]: True

[41]:
```
108==108 or 21 != 21
```

[41]: True

6 if— else => to handle single condition

7 if— elif— else => to handle Multiple condition

Observe in Python code: * if => statement in python * else => statement in python * : => colon => denotes start of if block i.e. any line written after colon belong to if condition * .... => see then as indentation i.e. 4 spaces => indentation indicates all code belong to only if and then another indentation indicates code for only else block

[42]:
```
# make variable with value as : money 100000

# see output of money > 2000
money=100000
money > 2000
```

[42]: True

[43]:
```
# assign money variable value of 10000
##### say you have this much ammount in your account

# start of if condition
# if money is greater then 1000 which is data science course free
# if money > 1000 is false i.e. you have less money then 1000 in your account␣
 ↪then else will work for now only if is working
if(money>1000):
  print("ds course free")
else:
  print("<1000 in account")
```

```
ds course free
```

[ ]:
```
# take a test_score variable with 80 in it.

# if test_score greater then 80 then print A grade
# elif test_score greater then 60 and less then 80 print B grade
# else print Nothing for you
```

[44]:
```
test_score=80
if(test_score>80):
  print("A")
elif test_score>60 and test_score< 80:
  print("B")
else:
  print("Nothing for you")
```

```
Nothing for you
```

8 Python Loops

[ ]:
```
"""
for iterating_variable in sequence:
    statement(s)
"""
```

[ ]:
```
'\nfor iterating_variable in sequence:\n    statement(s)\n'
```

[ ]:
```
for iterating_variable in range(10):
    print(iterating_variable)
```

```
0
1
2
3
4
5
6
7
8
9
```

[45]:
```
# print 'I love sports' 10 times using for loop
for i in range(10):
  print('I love sports')
```

```
I love sports
I love sports
I love sports
I love sports
```

```
I love sports
I love sports
I love sports
I love sports
I love sports
I love sports
```

10 => `stoping criteria` of, for loop

- `in` => keyword
- `sequence` => on which to itterate
- `:` => colon , start of for loop

`!=` = not equall to => `behaves as a stoping criteria`

[ ]:
```
# Syntax of while loop
"""
while comparison:
    statements(s)
"""
```

[ ]:
```
'\nwhile comparison:\n    statements(s)\n'
```

[48]:
```
# while loop

# save 0 in variable number

# print till 10 using while loop
n=0
while(n<10):
  print(n)
  n=n+1
```

```
0
1
2
3
4
5
6
7
8
9
```

- Initialized variable `number = 0` and then increment it's value in each iteration
- Loop will only continue to run only if value is less than 10

9 Type of Jump Statements

`Break Statement Continue Statement`

10 Break Statement

[52]:
```
# example that uses break statement in a for loop

# take range(10) and print 'The number is' + value
# break when num equals 5
for value in range(10):

  if(value==5):
    break
  print('The number is',value)
```

```
The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
```

11 Continue Statement

[ ]:
```
# Using same `for loop program` as in Break Statement section above
# Use a continue statement rather than a break statement

# take range(10) and print 'The number is' + value
# continue when num equals 5
```

[53]:
```
for value in range(10):

  if(value==5):
    continue
  print('The number is',value)
```

```
The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 6
The number is 7
The number is 8
The number is 9
```

12 String Manipulation

[ ]:
```
string_ = '' or "" or """ """
```

[54]:
```
# define a string variable with "We are creating next generation data science␣
 ↪eco-system at CollegeRanker"

str="We are creating next generation data science eco-system at CollegeRanker"
```

[55]:
```
# Find length of string including spaces

len(str)
```

[55]: 72

[56]:
```
# Access characters in a string with indexing i.e string[0]

str[4]
```

[56]: 'r'

[57]:
```
# Access characters with negative indexing i.e string[-1]

str[-2]
```

[57]: 'e'

13 String Slicing

[59]:
```
# select string from first to 6th element i.e string[:6]

str[:6]
```

[59]: 'We are'

[60]:
```
# select string from 7th to negative 10th element i.e string[7:-10]
str[7:-10]
```

[60]: 'creating next generation data science eco-system at Col'

Count of a particular `character` in a string

[61]:
```
str.count('a')
```

[61]: 7

Count of a particular `sub-string` in a string

[63]:
```
str.count('at')
```

[63]: 4

Find a substring in string using `find` and `index` function

[64]:
```
# .find() => if present it will return starting index, not found then it will return -1
# .index() => if present it will return starting index, not found then it will give error

str.find('are')
```

[64]: 3

[65]:
```
str.index('are')
```

[65]: 3

[66]:
```
### Checking whether string `startswith` or `endswith` a particular substring or not

str.startswith('We')
```

[66]: True

[68]:
```
str.endswith('CollegeRanker')
```

[68]: True

[69]:
```
### Converting string to upper case ###
str.upper()
```

[69]: 'WE ARE CREATING NEXT GENERATION DATA SCIENCE ECO-SYSTEM AT COLLEGERANKER'

[70]:
```
### Converting only first character of string to upper case

str.capitalize()
```

[70]: 'We are creating next generation data science eco-system at collegeranker'

[71]:
```
### Checking if string is in lower case or upper case

str.islower()
```

[71]: False

[72]:
```
str.isupper()
```

[72]: False

[90]:
```
### Checking if string is digit, alpabetic, alpha-numeric
str="ABCD"
str.lower().isalpha()
#print(str)
```

[90]: True

[91]:
```
str.isdigit()
```

[91]: False

[92]:
```
str.isalnum()
```

[92]: True

[93]:
```
# assign "C++ is easy to learn" to a new_str variable

new_str="C++ is easy to learn"
```

[95]:
```
### Replace C++ with Python
new_str.replace("C++","Python")
```

[95]: 'Python is easy to learn'

[96]:
```
### Use Split function on new_str ###

new_str.split(" ")
```

[96]: ['C++', 'is', 'easy', 'to', 'learn']

14 Python Functions

[ ]:
```
"""
def function_name():
    stetement(s)
"""
```

[ ]: '\ndef function_name():\n   stetement(s)\n'

[97]:
```
# define a function with welcome_message(name) and body 'Welcome to Functions !!
  ↪!'
def welcome():
  print('Welcome to Functions !!!')
welcome()
```

Welcome to Functions !!!

[98]:
```
# call a function with your name
def hi(name):
  print("HI",name)
name=input()
hi(name)
```

```
Akshay
HI Akshay
```

- `def` Keyword marking start of function
- `function name` to uniquely identify function
  - `function naming` follows same `rules of writing identifiers`
- `parameters`(arguments) to pass values to a function => totally optional
- `()` paranthesis
- `colon (:)` start of function
- `documentation string`(docstring) describe's what function does => totally optional
- `return statement` returns a value from function => totally optional
- inside colon is `function definition` it should always be present before function call or get an error

[101]:
```
# Write a function to add two number which are as 3 and 4
# in total variable store adition of 3 + 4
# print total variable

def total(n1,n2):
  total_variable=n1+n2
  return total_variable
print(total(3,4))
```

```
7
```

15 *Positional Arguments*

Most arguments are identified by their position in function call * Say `print(x,y)` will give different results from `print(y,x)`

What ever sequence is given while defining a function values must be taken in that sequence only * Otherwise use argument name **`(keyword arguments)`** to take values * We first define `positional argument` and then `keyword arguments`

[103]:
```
## Create substraction_function(small_number,large_number) and return␣
 ↪difference between large_number and small_number

def sub(s,l):
  diff=l-s
  return diff
print(sub(2,5))
```

```
3
```

[ ]:
```
# pass arguments in right order
```

[105]:
```
# always pass arguments using there name(keyword arguments) then order does not␣
 ↪matter

def sub(s,l):
  diff=l-s
  return diff
s=int(input())
l=int(input())
print(sub(s,l))
```

```
2
5
3
```

[ ]:
```

```

16 *Scope of Variables* means that part of program where we can access particular variable

- `Local Variable` => variables defined inside a function and can be only accessed from inside of that particular function
- `Global Variable` => variables defined outside a function and can be accessed throughout program

Let's define a global variable, `"global_variable"` outside function * We will return its value using a function `"randome_function"` and see that we would be able to access its value using that function also

[106]:
```
#### Observe every output from here onwords #####
# defining a global variable
global_variable = 'variable outside of function'

# defining function
def random_function():
    # accessing variable which is outside of this function
    return global_variable
```

[107]:
```
random_function()
```

[107]:
```
'variable outside of function'
```

`See we can acess the data of golbal variable from Inside of the Function`

17 => Let's see what will happen if we try to change value of global variable from Inside of the Function

[108]:
```
#### Observe every output from here onwords #####
# defining a global variable
global_variable = 'variable outside of function'

# defining function
def random_function():
    # changing value of global variable from inside of the function
    global_variable = 'changing variable outside of function from inside of␣
 ↪function'
    # accessing variable which is outside of this function
    return global_variable
```

[109]:
```
print(random_function())
print(global_variable)
```

```
changing variable outside of function from inside of function
variable outside of function
```

[ ]:

[ ]: