* Data types: integer, float, string,Boolean(whose value could be either True or False)
* Writing a Boolean type a variable:: variable\_name=True or False
* In Python, text in between quotes -- either single or double quotes -- is a string data type. An integer is a whole number, without a fraction, while a float is a real number that can contain a fractional part. When attempting to mix incompatible data types, you may encounter a TypeError. You can always check the data type of something using the type() function.
* You can use *type()* function to find out a data type .

print(type("a"))

output:: <class 'str'>

N.B:str=string

Take a look at another example:

print(type(2.5))

output:: <class 'float'>

* Python variables are case sensitive, so capitalization matters. Lowercase name, uppercase name and all caps name are all valid and different variable names
* Variables are names that we give to certain values in our programs. Those values can be of any data type; numbers, strings or even the results of operations
* The process of storing a value inside a variable is called *assignment*.
* An *expression* is a combination of numbers, symbols or other variables that produce a result when evaluated.
* you can name variables whatever you like but there are some restrictions.

1. First, you shouldn't use as variable names any of the key words or functions that Python reserves for its own, like print.
2. Variable names can't have any spaces and they must start with either a letter or an underscore.
3. Also, they can only be made up of letters, numbers and underscores

* I\_am\_a\_variable is the valid variable name.
* I\_am\_a\_variable2 is also a valid variable name.
* 1\_is\_a\_number is invalid because variable names must start with a letter or underscore.
* Apples\_&\_oranges is invalid because it uses the special character uppers hand.
* Adding two different type of data type:

Print(“a” +4) 🡺 this gives us error(string+integer)

Print(“a” + “b”) 🡺 no error(string + string)

Print(4+4.4) 🡺 this doesn’t give us any error(integer+float). Because computer automatically converts the integer 4 into float number and finally gives the result(8.4) in float. This process is called *Implicit conversion*, The interpreter automatically converts one data type into another. Implicit conversion is where the interpreter helps us out and automatically converts one data type into another, without having to explicitly tell it to do so.

By contrast, *explicit conversion* is where we manually convert from one data type to another by calling the relevant function for the data type we want to convert to

* What do you call a combination of numbers, symbols, or other values that produce a result when evaluated? Ans :: An explicit conversion
* An implicit conversion is when the interpreter automatically converts one data type to another.
* We can add a string and a number but before that we have to convert the number into string data type using str() function.

#to be able to add a string and a number, first we've to convert the number into string using str() function  
base=4  
height=7  
area=(1/2)\*base\*height  
print("the area of the triangle is :" + str(area))

* To convert an float to integer ::int(*float number*)
* To take input from user input() function used . If you want to print something while you are taking input from user :: input(“type something within quote with quotation marks:”)
* If you want to assign the inputted value into a variable:: name =input(“write something”)
* Syntax of if-else

If *conditions* :

Print(“write something here”)

else:

print(“write something here”)

* Code indentation is very important in python . Even if you don’t use indentation in right way at right place you will get indentation error

In the code below, you want to execute the third line when the “if ” condition is True. You have to indent the third line. That’s how python knows to execute the third line when the condition is True.

Price =7

If price < 10 :

Print (“I want that food”)

N.B:: If you don’t indent the line. You will get an indentation error.

Here are some example which show how to use indentation::

price = 7  
if price < 10:  
 print("I want that food")  
 print("I want the drinks")

output::

I want that food

I want the drinks

Comment: In this program, inside “if” block both print’s have same amount of indentation.

So interpreter will print both statement if the condition is true.

price = 7  
if price < 10:  
 print("I want that food")  
print("I want the drinks")

output:

I want that food

I want the drinks

Comment:: In this case, the third line is considered outside of the if block as it is not indented at all.

price = 7  
if price < 10:  
 print("I want that food")  
 print("I want the drinks")

output::

print("I want the drinks")

^

IndentationError: unexpected indent

Comment:: you have two options in your hand either you can indent both print statement same amount or don’t indent at all.

* < , > , =(one equal sign is used to assign a value to a variable) , ==(two equal signs are used to compare whether two value or two strings are equal) , != (not equal),//(floor division)
* If you want to add more conditions in the first line of if-else , then you have to put and and/or between all conditions
* FUNCTION(concept is same as fortran function)
* To define a function , we have to use *def* keyword

def greeting(name):

print(“welcome ”+ name)

greeting(“Donald Trump”)

1. def is the keyword to define a function
2. greeting is the function name
3. name is the parameter(also called argument)
4. print() is the body of the function(must be delimited by indentation)

* A function can have no parameters, or it can have multiple parameters. Parameters allow us to call a function and pass it data, with the data being available inside the function as variables with the same name as the parameters.
* After the colon, the function body starts. It’s important to note that in Python the function body is delimited by indentation. This means that all code indented to the right following a function definition is part of the function body. The first line that’s no longer indented is the boundary of the function body
* **Code Style::**

1. First off, you want your code to be self-documenting as possible. Self-documenting code is written in a way that's readable and doesn't conceal(hide) its intent. This principle can be applied to all aspects of writing code from picking your variable names to writing clear concise expressions.
2. Add comment

**Compare things**::  
>>>*print(10>1)*

*True*

>>>*print(“cat” == “dog”)*

*False*

>>>*print(1!=2)*

*True*

>>>*print(1 < “1”)*

*Traceback (most recent call last):*

*File “<stdin>”, line 1, in <module>*

*TypeError: ‘<’ not supported between instances of ‘int’ and ‘str’*

If you try to compare data types that aren’t compatible, like checking if a string is greater than an integer, Python will throw a ***TypeError***

>>>*print(1 == “1”)*

*False*

In this case , computer doesn’t send any error message because computer sees it as “no way a string can not be equal to a number” and it sends “your comparison is false”. But it can not compare whether a string is bigger or smaller than a number. That’s why it did send an error message

Logical Operator::

and, or, not

>>>*print(not 1>2)*

*True*

**Branching::**  
The ability of a program to alter its execution sequence is called branching.

The integer division is an operation between integers that yields two results which are both integers, the quotient and the remainder. So if we do an integer division between 5 and 2, the quotient is 2 and the remainder is 1. If we do an integer division between 11 and 3, the quotient is 3 and the remainder is 2.

>>>*def is\_even(number):*

*If number % 2 ==0:*

*return True*

*return False*

So how come we have these two return statements, one below the other, without an else statement? The trick is that when a return statement is executed, the function exits so that the code that follows doesn't get executed. This means that if the number is even, the computer will reach the return true statement and exit the function. Anything that comes after that will only be executed if the condition in the if statement was false.

def number\_group(number):

if number>0:

return "Positive"

elif number<0:

return "Negative"

else:

return "Zero"

print(number\_group(10)) #Should be Positive

print(number\_group(0)) #Should be Zero

print(number\_group(-5)) #Should be Negative

I have included this program to show how you can return a string from a function.

Conditionals Cheat Sheet

In earlier videos, we took a look at some of the built-in Python operators that allow us to compare values, and some logical operators we can use to combine values. We also learned how to use operators in if-else-elif blocks.

It’s a lot to learn but, with practice, it gets easier to remember it all. In the meantime, this handy cheat sheet gives you all the information you need at a glance.

Comparison operators

* a == b: a is equal to b
* a != b: a is different than b
* a < b: a is smaller than b
* a <= b: a is smaller or equal to b
* a > b: a is bigger than b
* a >= b: a is bigger or equal to b

Logical operators

* a and b: True if both a and b are True. False otherwise.
* a or b: True if either a or b or both are True. False if both are False.
* not a: True if a is False, False if a is True.