

Python

Python Variables:

Creating variables in python is easy, we just need to need to assign a value to a variable and it will take that type of the assigned value.

For example:

X=10, now X is an integer variable, there is no need for declaration of variable

Variable are case sensitive, so a=10, A='Deepthi', these 2 are treated as different variables.

To get the type of the variable we need to use type() function

Type(X) this will return integer as the type of variable X

String can be assigned by using double quotes or single quotes

Variable naming:

Rules of naming variables in python:

Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)
- A variable name cannot be any of the [Python keywords](#).

We can assign many values to multiple variables

For example:

X,y,z = 1,2,3

Here x is assigned with value 1, y is assigned with value 2 and z is assigned with value 3

one value to multiple variables

x,y,z = "Orange"

here x , y and z are assigned with a String value Orange

Unpack collection:

If we have collection of variables like list, tuple , we can assigned those to individual variables as well, for example

```
Fruits = ["apple", "Banana", "Orange"]
```

```
X,y,z= fruits
```

Then x is assigned with apple

Y is assigned with Banana

Z is assigned with Orange

Global variables:

A variable which is created outside a function is know as global variable which can used with in function as well as outside the function, its scope is global.

For example:

```
X=10
```

```
Def myfun():
```

```
    X=20
```

```
    Print(X)-----→ this print will print X value as 20
```

```
Myfun()
```

```
Print(X)-----→ this will print X value as 10 as it is global variable
```

To use a variable which is created with in function outside we need to use global variable while creating it, for example

```
Def myFunc():
```

```
    Global x
```

```
    X= 10
```

```
myFunc()
```

```
print(X) -----→ this print will orint X value as 10 even though it is within the function, because of global variable
```

Data Types:

Python has the following data types built-in by default, in these categories:

Text Type: str

Numeric Types: int, float, complex

Sequence Types: list, tuple, range

Mapping Type: dict

Set Types: set, frozenset

Boolean Type: bool

Binary Types: bytes, bytearray, memoryview

None Type: NoneType

There is no need to specify the data type when creating a variable, we will just assign a value and variable will take that datatype, for example:

X=10 ==> here X is integer data type

Y =" Hello" ==> Here Y is String datatype

Z = True ==> Z is Boolean data type

Python Numbers:

There are three numeric types in Python:

Int, float, complex

For example: x=10, y=10.5, z= 1j

To know what is the data type of these variable we need to use type function

Print(type(x)) => output will be integer

Int: Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

Float: Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

Complex: Complex numbers are written with a "j" as the imaginary part

We can convert from one type to another with the int(), float(), and complex() methods.

For example:

```
#convert from int to float:
x = float(1)

#convert from float to int:
y = int(2.8)

#convert from int to complex:
z = complex(1)

print(x)
print(y)
print(z)

print(type(x))
print(type(y))
print(type(z))
```

```
1.0
2
(1+0j)
<class 'float'>
<class 'int'>
<class 'complex'>
```

Random Number

Python does not have a random() function to make a random number, but Python has a built-in module called random that can be used to make random numbers:

```
import random

print(random.randrange(1, 10))
```

9

Python Strings

Strings in python are surrounded by either single quotation marks, or double quotation marks.

'hello' is the same as "hello".

You can use quotes inside a string, as long as they don't match the quotes surrounding the string: "He is called 'Johnny'"

Assigning a string to a variable is done with the variable name followed by an equal sign and the string: a = "Hello"

You can assign a multiline string to a variable by using three quotes:

```
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
```

Or

```
a = "Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."
```

```
a = "Hello, World!"  
print(a[1]) ==> this will print 0
```

To find the length of string we have to use len() function

Slicing

You can return a range of characters by using the slice syntax.

Specify the start index and the end index, separated by a colon, to return a part of the string.

```
b = "Hello, World!"  
print(b[2:5])
```

```
llo
```

Negative Indexing

Use negative indexes to start the slice from the end of the string:

```
b = "Hello, World!"  
print(b[-5:-2])
```

```
orl
```

Python has a set of built-in methods that you can use on strings.

Upper Case

The upper() method returns the string in upper case:

```
a = "Hello, World!"  
print(a.upper())
```

```
HELLO, WORLD!
```

Lower Case

The lower() method returns the string in lower case:

```
a = "Hello, World!"  
print(a.lower())
```

```
hello, world!
```

Remove Whitespace

Whitespace is the space before and/or after the actual text, and very often you want to remove this space.

The strip() method removes any whitespace from the beginning or the end:

```
a = " Hello, World! "  
print(a.strip())
```

```
Hello, World!
```

Replace String

The replace() method replaces a string with another string:

```
a = "Hello, World!"
print(a.replace("H", "J"))
```

```
Jello, World!
```

Split String

The `split()` method returns a list where the text between the specified separator becomes the list items.

```
a = "Hello, World!"
b = a.split(",")
print(b)
```

```
['Hello', ' World!']
```

String Concatenation

To concatenate, or combine, two strings you can use the `+` operator.

```
a = "Hello"
b = "World"
c = a + b
print(c)
```

```
HelloWorld
```

F-Strings

F-String was introduced in Python 3.6, and is now the preferred way of formatting strings.

To specify a string as an f-string, simply put an `f` in front of the string literal, and add curly brackets `{}` as placeholders for variables and other operations.

```
age = 36
txt = f"My name is John, I am {age}"
print(txt)
```

```
My name is John, I am 36
```

Escape Character

To insert characters that are illegal in a string, use an escape character.

An escape character is a backslash `\` followed by the character you want to insert.

```
txt = "We are the so-called \"Vikings\" from the north."
print(txt)
```

```
We are the so-called "Vikings" from the north.
```

Escape Characters

Other escape characters used in Python:

Code	Result
<code>\'</code>	Single Quote
<code>\\</code>	Backslash
<code>\n</code>	New Line
<code>\r</code>	Carriage Return
<code>\t</code>	Tab
<code>\b</code>	Backspace
<code>\f</code>	Form Feed
<code>\ooo</code>	Octal value
<code>\xhh</code>	Hex value

String Methods

Method	Description
<u>capitalize()</u>	Converts the first character to upper case
<u>casefold()</u>	Converts string into lower case
<u>center()</u>	Returns a centered string
<u>count()</u>	Returns the number of times a specified value occurs in a string
<u>encode()</u>	Returns an encoded version of the string
<u>endswith()</u>	Returns true if the string ends with the specified value
<u>expandtabs()</u>	Sets the tab size of the string
<u>find()</u>	Searches the string for a specified value and returns the position of where it was found
<u>format()</u>	Formats specified values in a string
<u>format_map()</u>	Formats specified values in a string
<u>index()</u>	Searches the string for a specified value and returns the position of where it was found
<u>isalnum()</u>	Returns True if all characters in the string are alphanumeric
<u>isalpha()</u>	Returns True if all characters in the string are in the alphabet
<u>isascii()</u>	Returns True if all characters in the string are ascii characters
<u>isdecimal()</u>	Returns True if all characters in the string are decimals
<u>isdigit()</u>	Returns True if all characters in the string are digits
<u>isidentifier()</u>	Returns True if the string is an identifier
<u>islower()</u>	Returns True if all characters in the string are lower case
<u>isnumeric()</u>	Returns True if all characters in the string are numeric
<u>isprintable()</u>	Returns True if all characters in the string are printable
<u>isspace()</u>	Returns True if all characters in the string are whitespaces
<u>istitle()</u>	Returns True if the string follows the rules of a title
<u>isupper()</u>	Returns True if all characters in the string are upper case
<u>join()</u>	Joins the elements of an iterable to the end of the string
<u>ljust()</u>	Returns a left justified version of the string
<u>lower()</u>	Converts a string into lower case
<u>lstrip()</u>	Returns a left trim version of the string

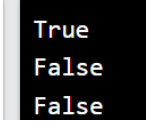
Boolean Values

In programming you often need to know if an expression is True or False.

You can evaluate any expression in Python, and get one of two answers, True or False.

When you compare two values, the expression is evaluated and Python returns the Boolean answer

```
print(10 > 9)  
print(10 == 9)  
print(10 < 9)
```



```
True  
False  
False
```