**What is a Programming Language and Python?**

Programming is a way for us to tell computers what to do. Computer is a very dumb machine and it only does what we tell it to do. Hence we learn programming and tell computers to do what we are very slow at - computation. If I ask you to calculate 5+6, you will immediately say 11. How about 23453453 X 56456?

You will start searching for a calculator or jump to a new tab to calculate the same.

**What is Python?**

* Python is an interpreted and a high-level programming language.
* It is a dynamically typed, general purpose programming language that supports an object-oriented programming approach as well as a functional programming approach.
* It was created by Guido Van Rossum in 1989 and released in 1991.

Uses of Python for Development.

Python is used in many application domains.

1. **Web development**

* Python offers many choices for [web development](http://wiki.python.org/moin/WebProgramming):
* Frameworks such as [Django](http://www.djangoproject.com/) and [Pyramid](http://www.pylonsproject.org/).
* Micro-frameworks such as [Flask](http://flask.pocoo.org/) and [Bottle](http://bottlepy.org/).
* Advanced content management systems such as [Plone](http://www.plone.org/) and [django CMS](https://www.django-cms.org/)

1. **Scientific and Numeric**

Python is widely used in [scientific and numeric](http://wiki.python.org/moin/NumericAndScientific) computing:

* [SciPy](http://scipy.org/) is a collection of packages for mathematics, science, and engineering.
* [Pandas](http://pandas.pydata.org/) is a data analysis and modeling library.
* [IPython](http://ipython.org/) is a powerful interactive shell that features easy editing and recording of a work session, and supports visualizations and parallel computing.
* Software development,
* Mathematics, Computing
* System scripting and Business applications.

**Features of Python**

* Python is simple and easy to understand.
* It is Interpreted and platform-independent which makes debugging very easy.
* Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
* Python is an open-source programming language.
* Python provides very big library support. Some of the popular libraries include NumPy, Tensorflow, Selenium, OpenCV, etc.
* It is possible to integrate other programming languages within python.
* The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.
* Python is an interpreted programming language, this means that as a developer you write Python (.py) files in a text editor and then put those files into the python interpreter to be executed.

**What is Python used for**

* Python is used in Data Visualization to create plots and graphical representations.
* Python helps in Data Analytics to analyze and understand raw data for insights and trends.
* It is used in AI and Machine Learning to simulate human behavior and to learn from past data without hard coding.
* It is used to create web applications.
* It can be used to handle databases.
* It is used in business and accounting to perform complex mathematical operations along with quantitative and qualitative analysis.

**Modules and pip in Python!**

Module is like a code library which can be used to borrow code written by somebody else in our python program. There are two types of modules in python:

1. Built in Modules - These modules are ready to import and use and ships with the python interpreter. there is no need to install such modules explicitly.
2. External Modules - These modules are imported from a third party file or can be installed using a package manager like pip or conda. Since this code is written by someone else, we can install different versions of a same module with time.

**The pip command**

It can be used as a package manager [pip](https://pip.pypa.io/en/stable/) to install a python module. Lets install a module called pandas using the following command

pip install pandas

**Using a module in Python (Usage)**

We use the import syntax to import a module in Python. Here is an example code:

import pandas

# Read and work with a file named 'words.csv'

df = pandas.read\_csv('words.csv')

print(df) # This will display first few rows from the words.csv file

Similarly we can install other modules and look into their documentations for usage instructions.

# Our First Program

Today we will write our first ever python program from scratch. It will consist of a bunch of print statements. print can be used to print something on the console in python

## Quick Quiz

Write a program to print a poem in Python. Choose the poem of your choice and publish your repl

print("---Your poem here---")

Please make sure you attempt this. Might be easy for some of you but please finish each and every task

# Comments, Escape sequence & Print in Python

Welcome to Day 5 of 100DaysOfCode. Today we will talk about Comments, Escape Sequences and little bit more about print statement in Python. We will also throw some light on Escape Sequences

# Python Comments

A comment is a part of the coding file that the programmer does not want to execute, rather the programmer uses it to either explain a block of code or to avoid the execution of a specific part of code while testing.

## Single-Line Comments:

To write a comment just add a ‘#’ at the start of the line.

### Example 1

#This is a 'Single-Line Comment'

print("This is a print statement.")

Output:

This is a print statement.

### Example 2

print("Hello World !!!") #Printing Hello World

Output:

Hello World !!!

**Variables and Data Types**

**What is a variable?**

Variable is like a container that holds data. Very similar to how our containers in kitchen holds sugar, salt etc Creating a variable is like creating a placeholder in memory and assigning it some value. In Python its as easy as writing:

a = 1

b = True

c = "Harry"

d = None

Copy

These are four variables of different data types.

**What is a Data Type?**

 Python Data Types are used to define the type of a variable. It defines what type of data we are going to store in a variable. The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters.

Python has various built-in data types which we will discuss with in this tutorial:

* Numeric - int, float, complex
* String - str
* Sequence - list, tuple, range
* Mapping - dict
* Boolean - bool
* Set - set
* None - NoneType

## Python Numeric Data Type

Python numeric data types store numeric values. Number objects are created when you assign a value to them. For example −

var1 = 1

var2 = 10

var3 = 10.023

Python supports four different numerical types −

* int (signed integers)
* long (long integers, they can also be represented in octal and hexadecimal)
* float (floating point real values)
* complex (complex numbers)

### Examples

Here are some examples of numbers −

|  |  |  |  |
| --- | --- | --- | --- |
| **Int** | **Long** | **float** | **complex** |
| 10 | 51924361L | 0.0 | 3.14j |
| 100 | -0x19323L | 15.20 | 45.j |
| -786 | 0122L | -21.9 | 9.322e-36j |
| 080 | 0xDEFABCECBDAECBFBAEl | 32.3+e18 | .876j |
| -0490 | 535633629843L | -90. | -.6545+0J |
| -0x260 | -052318172735L | -32.54e100 | 3e+26J |
| 0x69 | -4721885298529L | 70.2-E12 | 4.53e-7j |

|  |  |
| --- | --- |
| **Data type** | **Description** |
| bool\_ | Boolean (True or False) stored as a byte |
| int\_ | Default integer type (same as C long; normally either int64 or int32) |
| int8 | Byte (-128 to 127) |
| int16 | Integer (-32768 to 32767) |
| int32 | Integer (-2147483648 to 2147483647) |
| int64 | Integer (-9223372036854775808 to 9223372036854775807) |
| uint8 | Unsigned integer (0 to 255) |
| uint16 | Unsigned integer (0 to 65535) |
| uint32 | Unsigned integer (0 to 4294967295) |
| uint64 | Unsigned integer (0 to 18446744073709551615) |
| float\_ | Shorthand for float64. |
| float16 | Half precision float: sign bit, 5 bits exponent, 10 bits mantissa |
| float32 | Single precision float: sign bit, 8 bits exponent, 23 bits mantissa |
| float64 | Double precision float: sign bit, 11 bits exponent, 52 bits mantissa |
| complex\_ | Shorthand for complex128. |
| complex64 | Complex number, represented by two 32-bit floats |

* Python allows you to use a lowercase l with long, but it is recommended that you use only an uppercase L to avoid confusion with the number 1. Python displays long integers with an uppercase L.
* A complex number consists of an ordered pair of real floating-point numbers denoted by x + yj, where x and y are the real numbers and j is the imaginary unit.

### Example

Following is an example to show the usage of Integer, Float and Complex numbers:

# integer variable.

a=100

print("The type of variable having value", a, " is ", type(a))

# float variable.

b=20.345

print("The type of variable having value", b, " is ", type(b))

# complex variable.

c=10+3j

print("The type of variable having value", c, " is ", type(c))

d=True

print("The type of variable having value", d, " is ", type(d))

e= “Hello World”

print("The type of variable having value", e, " is ", type(e))

f=None

print("The type of variable having value", f, " is ", type(f))

The type of variable having value 100 is <class 'int'>

The type of variable having value 20.345 is <class 'float'>

The type of variable having value (10+3j) is <class 'complex'>

## Python String Data Type

Python Strings are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Subsets of strings can be taken using the slice operator ([ ] and [:] ) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

The plus (+) sign is the string concatenation operator and the asterisk (\*) is the repetition operator in Python. For example −

str = 'Hello World!'

print (str) # Prints complete string

print (str[0]) # Prints first character of the string

print (str[2:5]) # Prints characters starting from 3rd to 5th

print (str[2:]) # Prints string starting from 3rd character

print (str \* 2) # Prints string two times

print (str + "TEST") # Prints concatenated string

This will produce the following result −

Hello World!

H

llo

llo World!

Hello World!Hello World!

Hello World!TEST

## Python List Data Type

Python Lists are the most versatile compound data types. A Python list contains items separated by commas and enclosed within square brackets ([]). To some extent, Python lists are similar to arrays in C. One difference between them is that all the items belonging to a Python list can be of different data type where as C array can store elements related to a particular data type.

The values stored in a Python list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (\*) is the repetition operator. For example −

list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]

tinylist = [123, 'john']

print (list) # Prints complete list

print (list[0]) # Prints first element of the list

print (list[1:3]) # Prints elements starting from 2nd till 3rd

print (list[2:]) # Prints elements starting from 3rd element

print (tinylist \* 2) # Prints list two times

print (list + tinylist) # Prints concatenated lists

This produce the following result −

['abcd', 786, 2.23, 'john', 70.2]

abcd

[786, 2.23]

[2.23, 'john', 70.2]

[123, 'john', 123, 'john']

['abcd', 786, 2.23, 'john', 70.2, 123, 'john']

## Python Tuple Data Type

Python tuple is another sequence data type that is similar to a list. A Python tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.

The main differences between lists and tuples are: Lists are enclosed in brackets ( [ ] ) and their elements and size can be changed, while tuples are enclosed in parentheses ( ( ) ) and cannot be updated. Tuples can be thought of as **read-only** lists. For example −

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )

tinytuple = (123, 'john')

print (tuple) # Prints the complete tuple

print (tuple[0]) # Prints first element of the tuple

print (tuple[1:3]) # Prints elements of the tuple starting from 2nd till 3rd

print (tuple[2:]) # Prints elements of the tuple starting from 3rd element

print (tinytuple \* 2) # Prints the contents of the tuple twice

print (tuple + tinytuple) # Prints concatenated tuples

This produce the following result −

('abcd', 786, 2.23, 'john', 70.2)

abcd

(786, 2.23)

(2.23, 'john', 70.2)

(123, 'john', 123, 'john')

('abcd', 786, 2.23, 'john', 70.2, 123, 'john')

The following code is invalid with tuple, because we attempted to update a tuple, which is not allowed. Similar case is possible with lists −

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )

list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]

tuple[2] = 1000 # Invalid syntax with tuple

list[2] = 1000 # Valid syntax with list

## Python Ranges

Python **range()** is an in-built function in Python which returns a sequence of numbers starting from 0 and increments to 1 until it reaches a specified number.

We use **range()** function with for and while loop to generate a sequence of numbers. Following is the syntax of the function:

range(start, stop, step)

Here is the description of the parameters used:

* **start**: Integer number to specify starting position, (Its optional, Default: 0)
* **stop**: Integer number to specify starting position (It's mandatory)
* **step**: Integer number to specify increment, (Its optional, Default: 1)

### Examples

Following is a program which uses for loop to print number from 0 to 4 −

for i in range(5):

print(i)

This produce the following result −

0

1

2

3

4

Now let's modify above program to print the number starting from 1 instead of 0:

for i in range(1, 5):

print(i)

This produce the following result −

1

2

3

4

Again, let's modify the program to print the number starting from 1 but with an increment of 2 instead of 1:

for i in range(1, 5, 2):

print(i)

This produce the following result −

1

3

## Python Dictionary

Python dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]). For example −

dict = {}

dict['one'] = "This is one"

dict[2] = "This is two"

tinydict = {'name': 'john','code':6734, 'dept': 'sales'}

print (dict['one']) # Prints value for 'one' key

print (dict[2]) # Prints value for 2 key

print (tinydict) # Prints complete dictionary

print (tinydict.keys()) # Prints all the keys

print (tinydict.values()) # Prints all the values

This produce the following result −

This is one

This is two

{'dept': 'sales', 'code': 6734, 'name': 'john'}

['dept', 'code', 'name']

['sales', 6734, 'john']

Python dictionaries have no concept of order among elements. It is incorrect to say that the elements are "out of order"; they are simply unordered.

## Python Boolean Data Types

Python **boolean** type is one of built-in data types which represents one of the two values either **True** or **False**. Python **bool()** function allows you to evaluate the value of any expression and returns either True or False based on the expression.

### Examples

Following is a program which prints the value of boolean variables a and b −

a = True

# display the value of a

print(a)

# display the data type of a

print(type(a))

This produce the following result −

true

<class 'bool'>

Following is another program which evaluates the expressions and prints the return values:

# Returns false as a is not equal to b

a = 2

b = 4

print(bool(a==b))

# Following also prints the same

print(a==b)

# Returns False as a is None

a = None

print(bool(a))

# Returns false as a is an empty sequence

a = ()

print(bool(a))

# Returns false as a is 0

a = 0.0

print(bool(a))

# Returns false as a is 10

a = 10

print(bool(a))

This produce the following result −

False

False

False

False

False

True

## Python Data Type Conversion

Sometimes, you may need to perform conversions between the built-in data types. To convert data between different Python data types, you simply use the type name as a function.

### Conversion to int

Following is an example to convert number, float and string into integer data type:

a = int(1) # a will be 1

b = int(2.2) # b will be 2

c = int("3") # c will be 3

print (a)

print (b)

print (c)

This produce the following result −

1

2

3

### Conversion to float

Following is an example to convert number, float and string into float data type:

a = float(1) # a will be 1.0

b = float(2.2) # b will be 2.2

c = float("3.3") # c will be 3.3

print (a)

print (b)

print (c)

This produce the following result −

1.0

2.2

3.3

### Conversion to string

Following is an example to convert number, float and string into string data type:

a = str(1) # a will be "1"

b = str(2.2) # b will be "2.2"

c = str("3.3") # c will be "3.3"

print (a)

print (b)

print (c)

This produce the following result −

1

2.2

3.3

## Data Type Conversion Functions

There are several built-in functions to perform conversion from one data type to another. These functions return a new object representing the converted value.

# Python - Operators

Python operators are the constructs which can manipulate the value of operands. These are symbols used for the purpose of logical, arithmetic and various other operations. Consider the expression 4 + 5 = 9. Here, 4 and 5 are called **operands** and + is called **operator**.

## Types of Python Operators

Python language supports the following types of operators.

* Arithmetic Operators
* Comparison (Relational) Operators
* Assignment Operators
* Logical Operators
* Bitwise Operators

Let us have a quick look on all these operators one by one.

## Python Arithmetic Operators

Python arithmetic operators are used to perform mathematical operations on numerical values. These operations are Addition, Subtraction, Multiplication, Division, Modulus, Exponents and Floor Division.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Example** |
| + | Addition | 10 + 20 = 30 |
| - | Subtraction | 20 – 10 = 10 |
| \* | Multiplication | 10 \* 20 = 200 |
| / | Division | 20 / 10 = 2 |
| % | Modulus | 22 % 10 = 2 |
| \*\* | Exponent | 4\*\*2 = 16 |
| // | Floor Division | 9//2 = 4 |

### Example

Following is an example which shows all the above operations:

a = 21

b = 10

# Addition

print ("a + b : ", a + b)

# Subtraction

print ("a - b : ", a - b)

# Multiplication

print ("a \* b : ", a \* b)

# Division

print ("a / b : ", a / b)

# Modulus

print ("a % b : ", a % b)

# Exponent

print ("a \*\* b : ", a \*\* b)

# Floor Division

print ("a // b : ", a // b)

This produce the following result −

a + b : 31

a - b : 11

a \* b : 210

a / b : 2.1

a % b : 1

a \*\* b : 16679880978201

a // b : 2

## Python Comparison Operators

Python comparison operators compare the values on either sides of them and decide the relation among them. They are also called relational operators. These operators are equal, not equal, greater than, less than, greater than or equal to and less than or equal to.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Example** |
| == | Equal | 4 == 5 is not true. |
| != | Not Equal | 4 != 5 is true. |
| > | Greater Than | 4 > 5 is not true. |
| < | Less Than | 4 < 5 is true. |
| >= | Greater than or Equal to | 4 >= 5 is not true. |
| <= | Less than or Equal to | 4 <= 5 is true. |

### Example

Following is an example which shows all the above comparison operations:

a = 4

b = 5

# Equal

print ("a == b : ", a == b)

# Not Equal

print ("a != b : ", a != b)

# Greater Than

print ("a > b : ", a > b)

# Less Than

print ("a < b : ", a < b)

# Greater Than or Equal to

print ("a >= b : ", a >= b)

# Less Than or Equal to

print ("a <= b : ", a <= b)

This produce the following result −

a == b : False

a != b : True

a > b : False

a < b : True

a >= b : False

a <= b : True

## Python Assignment Operators

Python assignment operators are used to assign values to variables. These operators include simple assignment operator, addition assign, subtraction assign, multiplication assign, division and assign operators etc.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Example** |
| = | Assignment Operator | a = 10 |
| += | Addition Assignment | a += 5 (Same as a = a + 5) |
| -= | Subtraction Assignment | a -= 5 (Same as a = a - 5) |
| \*= | Multiplication Assignment | a \*= 5 (Same as a = a \* 5) |
| /= | Division Assignment | a /= 5 (Same as a = a / 5) |
| %= | Remainder Assignment | a %= 5 (Same as a = a % 5) |
| \*\*= | Exponent Assignment | a \*\*= 2 (Same as a = a \*\* 2) |
| //= | Floor Division Assignment | a //= 3 (Same as a = a // 3) |

### Example

Following is an example which shows all the above assignment operations:

# Assignment Operator

a = 10

# Addition Assignment

a += 5

print ("a += 5 : ", a)

# Subtraction Assignment

a -= 5

print ("a -= 5 : ", a)

# Multiplication Assignment

a \*= 5

print ("a \*= 5 : ", a)

# Division Assignment

a /= 5

print ("a /= 5 : ",a)

# Remainder Assignment

a %= 3

print ("a %= 3 : ", a)

# Exponent Assignment

a \*\*= 2

print ("a \*\*= 2 : ", a)

# Floor Division Assignment

a //= 3

print ("a //= 3 : ", a)

This produce the following result −

a += 5 : 105

a -= 5 : 100

a \*= 5 : 500

a /= 5 : 100.0

a %= 3 : 1.0

a \*\*= 2 : 1.0

a //= 3 : 0.0

## Python Logical Operators

There are following logical operators supported by Python language. Assume variable a holds 10 and variable b holds 20 then

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| and Logical AND | If both the operands are true then condition becomes true. | (a and b) is true. |
| or Logical OR | If any of the two operands are non-zero then condition becomes true. | (a or b) is true. |
| not Logical NOT | Used to reverse the logical state of its operand. | Not(a and b) is false. |

## Python Membership Operators

Python’s membership operators test for membership in a sequence, such as strings, lists, or tuples. There are two membership operators as explained below −

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| in | Evaluates to true if it finds a variable in the specified sequence and false otherwise. | x in y, here in results in a 1 if x is a member of sequence y. |
| not in | Evaluates to true if it does not finds a variable in the specified sequence and false otherwise. | x not in y, here not in results in a 1 if x is not a member of sequence y. |

## Python Bitwise Operators

Bitwise operator works on bits and performs bit by bit operation. Assume if a = 60; and b = 13; Now in the binary format their values will be 0011 1100 and 0000 1101 respectively. Following table lists out the bitwise operators supported by Python language with an example each in those, we use the above two variables (a and b) as operands −

a = 0011 1100

b = 0000 1101

--------------------------

a&b = 12 (0000 1100)

a|b = 61 (0011 1101)

a^b = 49 (0011 0001)

~a  = -61 (1100 0011)

a << 2 = 240 (1111 0000)

a>>2 = 15 (0000 1111)

There are following Bitwise operators supported by Python language

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Example** |
| & | Binary AND | Sets each bit to 1 if both bits are 1 |
| | | Binary OR | Sets each bit to 1 if one of two bits is 1 |
| ^ | Binary XOR | Sets each bit to 1 if only one of two bits is 1 |
| ~ | Binary Ones Complement | Inverts all the bits |
| << | Binary Left Shift | Shift left by pushing zeros in from the right and let the leftmost bits fall off |
| >> | Binary Right Shift | Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off |

### Example

Following is an example which shows all the above bitwise operations:

a = 60 # 60 = 0011 1100

b = 13 # 13 = 0000 1101

# Binary AND

c = a & b # 12 = 0000 1100

print ("a & b : ", c)

# Binary OR

c = a | b # 61 = 0011 1101

print ("a | b : ", c)

# Binary XOR

c = a ^ b # 49 = 0011 0001

print ("a ^ b : ", c)

# Binary Ones Complement

c = ~a; # -61 = 1100 0011

print ("~a : ", c)

# Binary Left Shift

c = a << 2; # 240 = 1111 0000

print ("a << 2 : ", c)

# Binary Right Shift

c = a >> 2; # 15 = 0000 1111

print ("a >> 2 : ", c)

This produce the following result −

a & b : 12

a | b : 61

a ^ b : 49

~a : -61

a >> 2 : 240

## a >> 2 : 15

# Python - Decision Making

Python programming language provides following types of decision making statements. Click the following links to check their detail.

|  |  |
| --- | --- |
| **Sr.No.** | **Statement & Description** |
| 1 | [if statements](https://www.tutorialspoint.com/python/python_if_statement.htm)  An **if statement** consists of a boolean expression followed by one or more statements. |
| 2 | [if...else statements](https://www.tutorialspoint.com/python/python_if_else.htm)  An **if statement** can be followed by an optional **else statement**, which executes when the boolean expression is FALSE. |
| 3 | [nested if statements](https://www.tutorialspoint.com/python/nested_if_statements_in_python.htm)  You can use one **if** or **else if** statement inside another **if** or **else if** statement(s). |

Let us go through each decision making briefly −

## Single Statement Suites

If the suite of an **if** clause consists only of a single line, it may go on the same line as the header statement.

#!/usr/bin/python

var = 100

if ( var = = 100 ) : print "Value of expression is 100"

print "Good bye!"

When the above code is executed, it produces the following result −

Value of expression is 100

Good bye!

### Syntax of if Else Statement

The syntax of the *if...else* statement is −

if expression:

statement(s)

else:

statement(s)

var1 = 100

if var1:

print "1 - Got a true expression value"

print var1

else:

print "1 - Got a false expression value"

print var1

var2 = 0

if var2:

print "2 - Got a true expression value"

print var2

else:

print "2 - Got a false expression value"

print var2

print "Good bye!"

## The *elif* Statement

The **elif** statement allows you to check multiple expressions for TRUE and execute a block of code as soon as one of the conditions evaluates to TRUE.

Similar to the **else**, the **elif** statement is optional. However, unlike **else**, for which there can be at most one statement, there can be an arbitrary number of **elif** statements following an **if**.

### syntax

if expression1:

statement(s)

elif expression2:

statement(s)

elif expression3:

statement(s)

else:

statement(s)

var = 100

if var = = 200:

print "1 - Got a true expression value"

print var

elif var = = 150:

print "2 - Got a true expression value"

print var

elif var = = 100:

print "3 - Got a true expression value"

print var

else:

print "4 - Got a false expression value"

print var

print "Good bye!"

## Multi-Line Comments:

To write multi-line comments you can use ‘#’ at each line or you can use the multiline string.

**Example 1:** The use of ‘#’.

#It will execute a block of code if a specified condition is true.

#If the condition is false then it will execute another block of code.

p = 7

if (p > 5):

print("p is greater than 5.")

else:

print("p is not greater than 5.")

Output:

p is greater than 5.

**Example 2:** The use of multiline string.

"""This is an if-else statement.

It will execute a block of code if a specified condition is true.

If the condition is false then it will execute another block of code."""

p = 7

if (p > 5):

print("p is greater than 5.")

else:

print("p is not greater than 5.")

### Output

p is greater than 5.

# Day 10 - Taking User Input in python

In python, we can take user input directly by using input() function.This input function gives a return value as string/character hence we have to pass that into a variable

## Syntax:

variable=input()

But input function returns the value as string. Hence we have to typecast them whenever required to another datatype.

## Example:

variable=int(input())

variable=float(input())

We can also display a text using input function. This will make input() function take user input and display a message as well

## Example:

a=input("Enter the name: ")

print(a)

## Output:

Enter the name: Harry

Harry

# Day 11 - What are strings?

In python, anything that you enclose between single or double quotation marks is considered a string. A string is essentially a sequence or array of textual data. Strings are used when working with Unicode characters.

## Example

name = "Harry"

print("Hello, " + name)

## Output

Hello, Harry

Note: It does not matter whether you enclose your strings in single or double quotes, the output remains the same.

Sometimes, the user might need to put quotation marks in between the strings. Example, consider the sentence: He said, “I want to eat an apple”.

How will you print this statement in python?: He said, "I want to eat an apple". We will definitely use single quotes for our convenience

print('He said, "I want to eat an apple".')

## Multiline Strings

If our string has multiple lines, we can create them like this:

a = """Lorem ipsum dolor sit amet,

consectetur adipiscing elit,

sed do eiusmod tempor incididunt

ut labore et dolore magna aliqua."""

print(a)

## Accessing Characters of a String

In Python, string is like an array of characters. We can access parts of string by using its index which starts from 0.  
Square brackets can be used to access elements of the string.

print(name[0])

print(name[1])

## Looping through the string

We can loop through strings using a for loop like this:

for character in name:

print(character)

Above code prints all the characters in the string name one by one!

# Day 12 - String Slicing & Operations on String

# Length of a String

We can find the length of a string using len() function.

## Example:

fruit = "Mango"

len1 = len(fruit)

print("Mango is a", len1, "letter word.")

Copy

## Output:

Mango is a 5 letter word

Copy

# String as an array

A string is essentially a sequence of characters also called an array. Thus we can access the elements of this array.

## Example:

pie = "ApplePie"

print(pie[:5])

print(pie[6]) #returns character at specified index

Copy

## Output:

Apple

i

Copy

Note: This method of specifying the start and end index to specify a part of a string is called slicing.

## Slicing Example:

pie = "ApplePie"

print(pie[:5]) #Slicing from Start

print(pie[5:]) #Slicing till End

print(pie[2:6]) #Slicing in between

print(pie[-8:]) #Slicing using negative index

Copy

## Output:

Apple

Pie

pleP

ApplePie

Copy

# Loop through a String:

Strings are arrays and arrays are iterable. Thus we can loop through strings.

## Example:

alphabets = "ABCDE"

for i in alphabets:

print(i)

## Output:

A

B

C

D

# Day 13 - String methods

Python provides a set of built-in methods that we can use to alter and modify the strings.

## upper() :

The upper() method converts a string to upper case.

### Example:

str1 = "AbcDEfghIJ"

print(str1.upper())

### Output:

ABCDEFGHIJ

## lower()

The lower() method converts a string to lower case.

### Example:

str1 = "AbcDEfghIJ"

print(str1.lower())

### Output:

abcdefghij

## strip() :

The strip() method removes any white spaces before and after the string.

### Example:

str2 = " Silver Spoon "

print(str2.strip)

### Output:

Silver Spoon

Hello

## replace() :

The replace() method replaces all occurences of a string with another string. Example:

str2 = "Silver Spoon"

print(str2.replace("Sp", "M"))

### Output:

Silver Moon

## split() :

The split() method splits the given string at the specified instance and returns the separated strings as list items.

### Example:

str2 = "Silver Spoon"

print(str2.split(" ")) #Splits the string at the whitespace " ".

### Output:

['Silver', 'Spoon']

We can use various other string methods to modify our strings.

## capitalize() :

The capitalize() method turns only the first character of the string to uppercase and the rest other characters of the string are turned to lowercase. The string has no effect if the first character is already uppercase.

### Example:

str1 = "hello"

capStr1 = str1.capitalize()

print(capStr1)

str2 = "hello WorlD"

capStr2 = str2.capitalize()

print(capStr2)

### Output:

Hello

Hello world

## center() :

The center() method aligns the string to the center as per the parameters given by the user.

### Example:

str1 = "Welcome to the Console!!!"

print(str1.center(50))

### Output:

Welcome to the Console!!!

We can also provide padding character. It will fill the rest of the fill characters provided by the user.

### Example:

str1 = "Welcome to the Console!!!"

print(str1.center(50, "."))

### Output:

............Welcome to the Console!!!.............

## count() :

The count() method returns the number of times the given value has occurred within the given string.

### Example:

str2 = "Abracadabra"

countStr = str2.count("a")

print(countStr)

### Output:

4

## endswith() :

The endswith() method checks if the string ends with a given value. If yes then return True, else return False.

### Example :

str1 = "Welcome to the Console !!!"

print(str1.endswith("!!!"))

### Output:

True

We can even also check for a value in-between the string by providing start and end index positions.

### Example:

str1 = "Welcome to the Console !!!"

print(str1.endswith("to", 4, 10))

### Output:

True

## find() :

The find() method searches for the first occurrence of the given value and returns the index where it is present. If given value is absent from the string then return -1.

### Example:

str1 = "He's name is Dan. He is an honest man."

print(str1.find("is"))

### Output:

10

As we can see, this method is somewhat similar to the index() method. The major difference being that index() raises an exception if value is absent whereas find() does not.

### Example:

str1 = "He's name is Dan. He is an honest man."

print(str1.find("Daniel"))

### Output:

-1

## index() :

The index() method searches for the first occurrence of the given value and returns the index where it is present. If given value is absent from the string then raise an exception.

### Example:

str1 = "He's name is Dan. Dan is an honest man."

print(str1.index("Dan"))

### Output:

13

As we can see, this method is somewhat similar to the find() method. The major difference being that index() raises an exception if value is absent whereas find() does not.

### Example:

str1 = "He's name is Dan. Dan is an honest man."

print(str1.index("Daniel"))

### Output:

ValueError: substring not found

## isalnum() :

The isalnum() method returns True only if the entire string only consists of A-Z, a-z, 0-9. If any other characters or punctuations are present, then it returns False.

### Example 1:

str1 = "WelcomeToTheConsole"

print(str1.isalnum())

Output:

True

## isalpha() :

The isalnum() method returns True only if the entire string only consists of A-Z, a-z. If any other characters or punctuations or numbers(0-9) are present, then it returns False.

### Example :

str1 = "Welcome"

print(str1.isalpha())

### Output:

True

## islower() :

The islower() method returns True if all the characters in the string are lower case, else it returns False.

### Example:

str1 = "hello world"

print(str1.islower())

### Output:

True

## isprintable() :

The isprintable() method returns True if all the values within the given string are printable, if not, then return False.

### Example :

str1 = "We wish you a Merry Christmas"

print(str1.isprintable())

### Output:

True

## isspace() :

The isspace() method returns True only and only if the string contains white spaces, else returns False.

### Example:

str1 = " " #using Spacebar

print(str1.isspace())

str2 = " " #using Tab

print(str2.isspace())

### Output:

True

True

## istitle() :

The istitile() returns True only if the first letter of each word of the string is capitalized, else it returns False.

### Example:

str1 = "World Health Organization"

print(str1.istitle())

### Output:

True

### Example:

str2 = "To kill a Mocking bird"

print(str2.istitle())

### Output:

False

## isupper() :

The isupper() method returns True if all the characters in the string are upper case, else it returns False.

### Example :

str1 = "WORLD HEALTH ORGANIZATION"

print(str1.isupper())

### Output:

True

## startswith() :

The endswith() method checks if the string starts with a given value. If yes then return True, else return False.

### Example :

str1 = "Python is a Interpreted Language"

print(str1.startswith("Python"))

### Output:

True

## swapcase() :

The swapcase() method changes the character casing of the string. Upper case are converted to lower case and lower case to upper case.

### Example:

str1 = "Python is a Interpreted Language"

print(str1.swapcase())

### Output:

pYTHON IS A iNTERPRETED lANGUAGE

### title() :

The title() method capitalizes each letter of the word within the string.

### Example:

str1 = "He's name is Dan. Dan is an honest man."

print(str1.title())

### Output:

He'S Name Is Dan. Dan Is An Honest Man.

# Day 16 - Match Case Statements

To implement switch-case like characteristics very similar to if-else functionality, we use a match case in python. If you are coming from a C, C++ or Java like language, you must have heard of switch-case statements. If this is your first language, dont worry as I will tell you everything you need to know about match case statements in this video!

A match statement will compare a given variable’s value to different shapes, also referred to as the pattern. The main idea is to keep on comparing the variable with all the present patterns until it fits into one.

The match case consists of three main entities :

1. The match keyword
2. One or more case clauses
3. Expression for each case

The case clause consists of a pattern to be matched to the variable, a condition to be evaluated if the pattern matches, and a set of statements to be executed if the pattern matches.

## Syntax:

match variable\_name:

case ‘pattern1’ : //statement1

case ‘pattern2’ : //statement2

…

case ‘pattern n’ : //statement n

### Example:

x = 4

# x is the variable to match

match x:

# if x is 0

case 0:

print("x is zero")

# case with if-condition

case 4 if x % 2 == 0:

print("x % 2 == 0 and case is 4")

# Empty case with if-condition

case \_ if x < 10:

print("x is < 10")

# default case(will only be matched if the above cases were not matched)

# so it is basically just an else:

case \_:

print(x)

### Match Case in Python

### Python offers a simple and effective way to test multiple values and perform [conditional actions:](https://www.udacity.com/blog/2021/07/how-to-write-your-first-python-application.html) the match-case statement. In case you’re familiar with C++, it works similarly to the switch case:

# First, ask the player about their CPU

cpuModel = str.lower(input("Please enter your CPU model: "))

# The match statement evaluates the variable's value

match cpuModel:

case "celeron": # We test for different values and print different messages

        print ("Forget about it and play Minesweeper instead...")

case "core i3":

        print ("Good luck with that ;)")

case "core i5":

        print ("Yeah, you should be fine.")

case "core i7":

        print ("Have fun!")

case "core i9":

        print ("Our team designed nice loading screens… Too bad you won't see them...")

case \_: # the underscore character is used as a catch-all.

        print ("Is that even a thing?")

**Checking entered character vowel or not using Match Case in Python**

# Enter character

ch = input("Please enter character: ")

# The match statement evaluates the variable's value

match ch:

        case "a":

                print ("Wovel :",ch)

        case "e":

                print ("Wovel :",ch)

        case "i":

                print ("Wovel ", ch)

        case "o":

                print ("Wovel ", ch)

        case "u":

                print ("Wovel ", ch)

        case \_:

                print ("Is that even a thing?")

x % 2 == 0 and case is 4

**Day 17 - Introduction to Loops**

Sometimes a programmer wants to execute a group of statements a certain number of times. This can be done using loops. Based on this loops are further classified into following main types;

* for loop
* while loop

**The for Loop**

for loops can iterate over a sequence of iterable objects in python. Iterating over a sequence is nothing but iterating over strings, lists, tuples, sets and dictionaries.

**Example: iterating over a string:**

name = 'Abhishek'

for i in name:

print(i, end=", ")

Copy

**Output:**

A, b, h, i, s, h, e, k,

Copy

**Example: iterating over a list:**

colors = ["Red", "Green", "Blue", "Yellow"]

for x in colors:

print(x)

Copy

**Output:**

Red

Green

Blue

Yellow

Copy

Similarly, we can use loops for lists, sets and dictionaries.

**range():**

What if we do not want to iterate over a sequence? What if we want to use for loop for a specific number of times?

Here, we can use the range() function.

**Example:**

for k in range(5):

print(k)

Copy

**Output:**

0

1

2

3

4

Copy

**Here, we can see that the loop starts from 0 by default and increments at each iteration.**

But we can also loop over a specific range.

**Example:**

for k in range(4,9):

print(k)

Copy

**Output:**

4

5

6

7

8

# Day 18 - Python While Loop

As the name suggests, while loops execute statements while the condition is True. As soon as the condition becomes False, the interpreter comes out of the while loop.

## Example:

count = 5

while (count > 0):

print(count)

count = count - 1

Copy

## Output:

5

4

3

2

1

Copy

Here, the count variable is set to 5 which decrements after each iteration. Depending upon the while loop condition, we need to either increment or decrement the counter variable (the variable count, in our case) or the loop will continue forever.

# Else with While Loop

We can even use the else statement with the while loop. Essentially what the else statement does is that as soon as the while loop condition becomes False, the interpreter comes out of the while loop and the else statement is executed.

## Example:

x = 5

while (x > 0):

print(x)

x = x - 1

else:

print('counter is 0')

## Output:

5

4

3

2

1

counter is 0

# Day 19 - Break & Continue Statement

## Break Statement

The break statement enables a program to skip over a part of the code. A break statement terminates the very loop it lies within.

## example

for i in range(1,101,1):

print(i ,end=" ")

if(i==50):

break

else:

print("Mississippi")

print("Thank you")

Copy

### output

1 Mississippi

2 Mississippi

3 Mississippi

4 Mississippi

5 Mississippi

.

.

.

50 Mississippi

Copy

## Continue Statement

The continue statement skips the rest of the loop statements and causes the next iteration to occur.

## example

for i in [2,3,4,6,8,0]:

if (i%2!=0):

continue

print(i)

Copy

## output

2

4

6

8

0

# Day 20 - Python Functions

A function is a block of code that performs a specific task whenever it is called. In bigger programs, where we have large amounts of code, it is advisable to create or use existing functions that make the program flow organized and neat.

There are two types of functions:

1. Built-in functions
2. User-defined functions

## Built-in functions:

These functions are defined and pre-coded in python. Some examples of built-in functions are as follows:

min(), max(), len(), sum(), type(), range(), dict(), list(), tuple(), set(), print(), etc.

## User-defined functions:

We can create functions to perform specific tasks as per our needs. Such functions are called user-defined functions.

### Syntax:

def function\_name(parameters):

pass

# Code and Statements

Copy

* Create a function using the def keyword, followed by a function name, followed by a paranthesis (()) and a colon(:).
* Any parameters and arguments should be placed within the parentheses.
* Rules to naming function are similar to that of naming variables.
* Any statements and other code within the function should be indented.

### Calling a function:

We call a function by giving the function name, followed by parameters (if any) in the parenthesis.

Example:

def name(fname, lname):

print("Hello,", fname, lname)

name("Sam", "Wilson")

Copy

Output:

Hello, Sam Wilson

# Day 21 - Function Arguments and Return Statement

There are four types of arguments that we can provide in a function:

* Default Arguments
* Keyword Arguments
* Variable length Arguments
* Required Arguments

### Default arguments:

We can provide a default value while creating a function. This way the function assumes a default value even if a value is not provided in the function call for that argument.

Example:

def name(fname, mname = "Jhon", lname = "Whatson"):

print("Hello,", fname, mname, lname)

name("Amy")

Copy

Output:

Hello, Amy Jhon Whatson

Copy

### Keyword arguments:

We can provide arguments with key = value, this way the interpreter recognizes the arguments by the parameter name. Hence, the the order in which the arguments are passed does not matter.

Example:

def name(fname, mname, lname):

print("Hello,", fname, mname, lname)

name(mname = "Peter", lname = "Wesker", fname = "Jade")

Copy

Output:

Hello, Jade Peter Wesker

Copy

### Required arguments:

In case we don’t pass the arguments with a key = value syntax, then it is necessary to pass the arguments in the correct positional order and the number of arguments passed should match with actual function definition.

Example 1: when number of arguments passed does not match to the actual function definition.

def name(fname, mname, lname):

print("Hello,", fname, mname, lname)

name("Peter", "Quill")

Copy

Output:

name("Peter", "Quill")\

TypeError: name() missing 1 required positional argument: 'lname'

Copy

Example 2: when number of arguments passed matches to the actual function definition.

def name(fname, mname, lname):

print("Hello,", fname, mname, lname)

name("Peter", "Ego", "Quill")

Copy

Output:

Hello, Peter Ego Quill

Copy

### Variable-length arguments:

Sometimes we may need to pass more arguments than those defined in the actual function. This can be done using variable-length arguments.

There are two ways to achieve this:

#### Arbitrary Arguments:

While creating a function, pass a \* before the parameter name while defining the function. The function accesses the arguments by processing them in the form of tuple.

Example:

def name(\*name):

print("Hello,", name[0], name[1], name[2])

name("James", "Buchanan", "Barnes")

Copy

Output:

Hello, James Buchanan Barnes

Copy

#### Keyword Arbitrary Arguments:

While creating a function, pass a \* before the parameter name while defining the function. The function accesses the arguments by processing them in the form of dictionary.

Example:

def name(\*\*name):

print("Hello,", name["fname"], name["mname"], name["lname"])

name(mname = "Buchanan", lname = "Barnes", fname = "James")

Copy

Output:

Hello, James Buchanan Barnes

Copy

# return Statement

The return statement is used to return the value of the expression back to the calling function.

Example:

def name(fname, mname, lname):

return "Hello, " + fname + " " + mname + " " + lname

print(name("James", "Buchanan", "Barnes"))

Output:

Hello, James Buchanan Barnes

**Day 22 - Python Lists**

* Lists are ordered collection of data items.
* They store multiple items in a single variable.
* List items are separated by commas and enclosed within square brackets [].
* Lists are changeable meaning we can alter them after creation.

Example 1:

lst1 = [1,2,2,3,5,4,6]

lst2 = ["Red", "Green", "Blue"]

print(lst1)

print(lst2)

Output:

[1, 2, 2, 3, 5, 4, 6]

['Red', 'Green', 'Blue']

Example 2:

details = ["Abhijeet", 18, "FYBScIT", 9.8]

print(details)

Output:

['Abhijeet', 18, 'FYBScIT', 9.8]

# Day 23 - List Methods

## list.sort()

This method sorts the list in ascending order. The original list is updated

### Example 1:

colors = ["violet", "indigo", "blue", "green"]

colors.sort()

print(colors)

num = [4,2,5,3,6,1,2,1,2,8,9,7]

num.sort()

print(num)

### Output:

['blue', 'green', 'indigo', 'violet']\

[1, 1, 2, 2, 2, 3, 4, 5, 6, 7, 8, 9]

What if you want to print the list in descending order?  
We must give reverse=True as a parameter in the sort method.

### Example:

colors = ["violet", "indigo", "blue", "green"]

colors.sort(reverse=True)

print(colors)

num = [4,2,5,3,6,1,2,1,2,8,9,7]

num.sort(reverse=True)

print(num)

#### Output:

['violet', 'indigo', 'green', 'blue']

[9, 8, 7, 6, 5, 4, 3, 2, 2, 2, 1, 1]

The reverse parameter is set to False by default.

Note: Do not mistake the reverse parameter with the reverse method.

## reverse()

This method reverses the order of the list.

#### Example:

colors = ["violet", "indigo", "blue", "green"]

colors.reverse()

print(colors)

num = [4,2,5,3,6,1,2,1,2,8,9,7]

num.reverse()

print(num)

#### Output:

['green', 'blue', 'indigo', 'violet']

[7, 9, 8, 2, 1, 2, 1, 6, 3, 5, 2, 4]

## index()

This method returns the index of the first occurrence of the list item.

#### Example:

colors = ["violet", "green", "indigo", "blue", "green"]

print(colors.index("green"))

num = [4,2,5,3,6,1,2,1,3,2,8,9,7]

print(num.index(3))

Output:

1

3

## count()

Returns the count of the number of items with the given value.

#### Example:

colors = ["violet", "green", "indigo", "blue", "green"]

print(colors.count("green"))

num = [4,2,5,3,6,1,2,1,3,2,8,9,7]

Copy

#### Output:

2

3

Copy

## copy()

Returns copy of the list. This can be done to perform operations on the list without modifying the original list.

#### Example:

colors = ["violet", "green", "indigo", "blue"]

newlist = colors.copy()

print(colors)

print(newlist)

Copy

#### Output:

['violet', 'green', 'indigo', 'blue']

['violet', 'green', 'indigo', 'blue']

Copy

## append():

This method appends items to the end of the existing list.

#### Example:

colors = ["violet", "indigo", "blue"]

colors.append("green")

print(colors)

Copy

#### Output:

['violet', 'indigo', 'blue', 'green']

Copy

## insert():

This method inserts an item at the given index. User has to specify index and the item to be inserted within the insert() method.

#### Example:

colors = ["violet", "indigo", "blue"]

# [0] [1] [2]

colors.insert(1, "green") #inserts item at index 1

# updated list: colors = ["violet", "green", "indigo", "blue"]

# indexs [0] [1] [2] [3]

print(colors)

Copy

#### Output:

['violet', 'green', 'indigo', 'blue']

Copy

## extend():

This method adds an entire list or any other collection datatype (set, tuple, dictionary) to the existing list.

#### Example 1:

#add a list to a list

colors = ["violet", "indigo", "blue"]

rainbow = ["green", "yellow", "orange", "red"]

colors.extend(rainbow)

print(colors)

Copy

#### Output:

['violet', 'indigo', 'blue', 'green', 'yellow', 'orange', 'red']

Copy

## Concatenating two lists:

You can simply concatenate two lists to join two lists.

#### Example:

colors = ["violet", "indigo", "blue", "green"]

colors2 = ["yellow", "orange", "red"]

print(colors + colors2)

Copy

#### Output:

['violet', 'indigo', 'blue', 'green', 'yellow', 'orange', 'red']

# Day 24 - Python Tuples & Tuple Indexes

Tuples are ordered collection of data items. They store multiple items in a single variable. Tuple items are separated by commas and enclosed within round brackets (). Tuples are unchangeable meaning we can not alter them after creation.

### Example 1:

tuple1 = (1,2,2,3,5,4,6)

tuple2 = ("Red", "Green", "Blue")

print(tuple1)

print(tuple2)

Copy

### Output:

(1, 2, 2, 3, 5, 4, 6)

('Red', 'Green', 'Blue')

Copy

### Example 2:

details = ("Abhijeet", 18, "FYBScIT", 9.8)

print(details)

Copy

### Output:

('Abhijeet', 18, 'FYBScIT', 9.8)

Copy

## Tuple Indexes

Each item/element in a tuple has its own unique index. This index can be used to access any particular item from the tuple. The first item has index [0], second item has index [1], third item has index [2] and so on.

### Example:

country = ("Spain", "Italy", "India",)

# [0] [1] [2]

Copy

## Accessing tuple items:

### I. Positive Indexing:

As we have seen that tuple items have index, as such we can access items using these indexes.

### Example:

country = ("Spain", "Italy", "India",)

# [0] [1] [2]

print(country[0])

print(country[1])

print(country[2])

Copy

### Output:

Spain

Italy

India

Copy

### II. Negative Indexing:

Similar to positive indexing, negative indexing is also used to access items, but from the end of the tuple. The last item has index [-1], second last item has index [-2], third last item has index [-3] and so on.

### Example:

country = ("Spain", "Italy", "India", "England", "Germany")

# [0] [1] [2] [3] [4]

print(country[-1]) # Similar to print(country[len(country) - 1])

print(country[-3])

print(country[-4])

Copy

### Output:

Germany

India

Italy

Copy

## III. Check for item:

We can check if a given item is present in the tuple. This is done using the in keyword.

### Example 1:

country = ("Spain", "Italy", "India", "England", "Germany")

if "Germany" in country:

print("Germany is present.")

else:

print("Germany is absent.")

Copy

### Output:

Germany is present.

Copy

Germany is present.

### Example 2:

country = ("Spain", "Italy", "India", "England", "Germany")

if "Russia" in country:

print("Russia is present.")

else:

print("Russia is absent.")

Copy

### Output:

Russia is absent.

Copy

### IV. Range of Index:

You can print a range of tuple items by specifying where do you want to start, where do you want to end and if you want to skip elements in between the range.

### Syntax:

Tuple[start : end : jumpIndex]

Copy

Note: jump Index is optional. We will see this in given examples.

### Example: Printing elements within a particular range:

animals = ("cat", "dog", "bat", "mouse", "pig", "horse", "donkey", "goat", "cow")

print(animals[3:7]) #using positive indexes

print(animals[-7:-2]) #using negative indexes

Copy

### Output:

('mouse', 'pig', 'horse', 'donkey')

('bat', 'mouse', 'pig', 'horse', 'donkey')

Copy

Here, we provide index of the element from where we want to start and the index of the element till which we want to print the values. Note: The element of the end index provided will not be included.

### Example: Printing all element from a given index till the end

animals = ("cat", "dog", "bat", "mouse", "pig", "horse", "donkey", "goat", "cow")

print(animals[4:]) #using positive indexes

print(animals[-4:]) #using negative indexes

Copy

### Output:

('pig', 'horse', 'donkey', 'goat', 'cow')

('horse', 'donkey', 'goat', 'cow')

Copy

When no end index is provided, the interpreter prints all the values till the end.

### Example: printing all elements from start to a given index

animals = ("cat", "dog", "bat", "mouse", "pig", "horse", "donkey", "goat", "cow")

print(animals[:6]) #using positive indexes

print(animals[:-3]) #using negative indexes

Copy

### Output:

('cat', 'dog', 'bat', 'mouse', 'pig', 'horse')

('cat', 'dog', 'bat', 'mouse', 'pig', 'horse')

Copy

When no start index is provided, the interpreter prints all the values from start up to the end index provided.

### Example: Print alternate values

animals = ("cat", "dog", "bat", "mouse", "pig", "horse", "donkey", "goat", "cow")

print(animals[::2]) #using positive indexes

print(animals[-8:-1:2]) #using negative indexes

Copy

### Output:

('cat', 'bat', 'pig', 'donkey', 'cow')

('dog', 'mouse', 'horse', 'goat')

Copy

Here, we have not provided start and end index, which means all the values will be considered. But as we have provided a jump index of 2 only alternate values will be printed.

### Example: printing every 3rd consecutive withing given range

animals = ("cat", "dog", "bat", "mouse", "pig", "horse", "donkey", "goat", "cow")

print(animals[1:8:3])

Copy

### Output:

('dog', 'pig', 'goat')

# Day 25 - Manipulating Tuples & Tuple Methods

## Manipulating Tuples

Tuples are immutable, hence if you want to add, remove or change tuple items, then first you must convert the tuple to a list. Then perform operation on that list and convert it back to tuple.

#### Example:

countries = ("Spain", "Italy", "India", "England", "Germany")

temp = list(countries)

temp.append("Russia") #add item

temp.pop(3) #remove item

temp[2] = "Finland" #change item

countries = tuple(temp)

print(countries)

Copy

#### Output:

('Spain', 'Italy', 'Finland', 'Germany', 'Russia')

Copy

Thus, we convert the tuple to a list, manipulate items of the list using list methods, then convert list back to a tuple.

However, we can directly concatenate two tuples without converting them to list.

#### Example:

countries = ("Pakistan", "Afghanistan", "Bangladesh", "ShriLanka")

countries2 = ("Vietnam", "India", "China")

southEastAsia = countries + countries2

print(southEastAsia)

Copy

#### Output:

('Pakistan', 'Afghanistan', 'Bangladesh', 'ShriLanka', 'Vietnam', 'India', 'China')

Copy

## Tuple methods

As tuple is an immutable type of collection of elements it has limited built-in methods. They are explained below

## count() Method

The count() method of Tuple returns the number of times the given element appears in the tuple.

### Syntax:

tuple.count(element)

Copy

### Example:

Tuple1 = (0, 1, 2, 3, 2, 3, 1, 3, 2)

res = Tuple1.count(3)

print('Count of 3 in Tuple1 is:', res)

Copy

### Output:

3

Copy

# index() method

The Index() method returns the first occurrence of the given element from the tuple.

### Syntax:

tuple.index(element, start, end)

Copy

Note: This method raises a ValueError if the element is not found in the tuple.

### Example:

Tuple = (0, 1, 2, 3, 2, 3, 1, 3, 2)

res = Tuple.index(3)

print('First occurrence of 3 is', res)

Copy

#### Output

3