# Data Structures

* are a way of organizing data so that it can be accessed more efficiently depending upon the situation.
* are fundamentals of any programming language around which a program is built.
* Python helps to learn the fundamental of these data structures in a simpler way as compared to other programming languages.

# Built-in Data Structures

# Lists

* + are used to store multiple items in a single variable.
  + are just like the arrays, declared in other languages which is an ordered collection of data.
  + Python lists are very flexible as the items in a list do not need to be of the same type.
  + List items are ordered, changeable, and allow duplicate values.
  + are created using square brackets; it's also possible to use the list() constructor.
  + are indexed, the first item has index [0], the second item has index [1] etc.
  + The len() function is used to determine the number of items a list has.
  + In Python, lists are defined as objects with the data type 'list'. The type() function determines the data type of a list.

## Example:

thislist = ["apple", 5, True, 2.5, 5] # OR thislist = list(("apple", 5, True, 2.5, 5))

print(thislist) # prints ["apple", 5, True, 2.5, 5]

print(len(thislist)) # prints 5

print(type(mylist)) # prints <class 'list'>

## Accessing Items

* + By referring to the index number: print(list[3]) #Prints the 4th item
  + By negative indexing: print(list[-1]) #Prints the last item, and the 2nd last item if [-2]
  + By defining the start (included) and end (not included) of a range, we can specify a range of indexes:  
    print(list[2:5]) #Prints the 3rd, 4th & 5th item
  + By leaving out the start value, the range will start at the first item:  
    print(list[:5] #Prints the items from the start to, but NOT including, the 5th item)
  + By leaving out the end value, the range will go on to the end of the list:  
    print(list[2:] #Prints from the 3rd item to, including, the end)
  + By specifying negative indexes if we want to start the search from the end of the list:  
    print(list[-4:-1]) #Prints the items from index -4 (included) to index -1 (last item, excluded)

### Example

thislist = ["apple", 5, True, 2.5, 5]

print(thislist[2]) # prints True

print(thislist[-2]) # prints 2.5

print(thislist[1:4]) # prints [5, True, 2.5]

print(thislist[:3]) # prints ["apple", 5, True]

print(thislist[1:]) # prints [5, True, 2.5, 5]

print(thislist[-4:-1]) # prints [5, True, 2.5]

# Variables

* Python has no command for declaring a variable. It is created when the first value is assigned.
* Multiple values can be assigned to multiple variables in one line. Eg., x, y, z = "Orange", "Banana", "Cherry"
* Same value can be assignd to multiple variables. Eg., x = y = z = "Orange"
* The Python print() function is often used to output variables.
* Variables that are created outside of a function are known as **Global Variables**.
* A variable can have a short name like x and y or a more descriptive name age, carname, total\_volume.
  + **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. Eg.,a = 4 A = "Hanna" # A will not overwrite a

### Example:

# Legal variable names with different techniques & values

myVar = 5 # camel case with int data type

MyVar = "Hanna" # pascal case with str data type

my\_var = 'Ruth' # snake case with str data type

\_x = 5.7 # float

x2 = "Tsige"

# Illegal variable names

2x = "Hanna"

x- = 'Ruth'

x 2 = "Tsige"

## Unpack a Collection

* If there is a collection of values in a list, tuple etc. Python allows to extract the values into variables. This is called Unpacking.

### Example:

fruits = ["apple", "banana", "cherry"]

x, y, z = fruits

print(x) # apple

print(y) # banana

print(z) # cherry

## Casting

* It's used to specify the data type of a variable.

### Example:

x = str(3) # x will be '3'

y = int(3) # y will be 3

z = float(3) # z will be 3.0

## **type()** Function

* It's used to get the data type of a variable.

### Example:

x = 5

y = "Hanna"

print(type(x)) # Prints <class 'int'>

print(type(y)) # Prints <class 'str'>

## Output Variables

### Example 1:

x = "Python is awesome"

print(x) # Prints Python is awesome

### Example 2:

# Multiple variables separated by a comma:

x = "Python"

y = "is"

z = "awesome"

print(x, y, z) # Prints Python is awesome

* In the print() function, combining a string and a number with the + operator, Python will give you an error:
* The best way to output multiple variables in the print() function is to separate them with commas, which even support different data types:

## Global Variables

* They can be used by everyone, both inside of functions and outside.

### Example :

* Create a variable outside of a function, and use it inside the function

x = "awesome"

def myfunc():

print("Python is " + x) # Prints Python is awesome

myfunc()

### **global** Keyword

* Normally, when a variable is created inside a function, that variable is local, and can only be used inside that function.
* It's used to create a global variable inside a function.

#### Example 1:

* If the global keyword is used, the variable belongs to the global scope:

def myfunc():

global x

x = "fantastic"

myfunc()

print("Python is " + x) # Prints Python is fantastic.

#### Example 2:

* Also, use the global keyword to refer to the variable when changing the value of a global variable inside a function:

x = "awesome"

def myfunc():

global x

x = "fantastic"

myfunc()

print("Python is " + x) # Prints Python is fantastic.

# Data Types

* It is an important concept.
* Variables can store data of different types, and different types can do different things.
* How to get the data type is found in the **type() function** example.
* Setting the data type is found in the very first example.

## Built-in Data Types:

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

## Setting the Specific Data Type

### Example

x = str("Hanna")x = int(20)x = float(20.5)x = complex(1j)x = list(("apple", "banana", "cherry"))x = tuple(("apple", "banana", "cherry"))