# PYTHON-BASICS

Data Types

# Python-Data types

- 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-built-in data types

| |
|---|
| Numeric - int, float, complex |
| String - str |
| Sequence - list, tuple, range |
| Binary - bytes, bytearray, memoryview |
| Mapping - dict |
| Boolean - bool |
| Set - set, frozenset |
| 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)

# Example

```
# integer variable.
a=169
print("The type of variable having value", a, " is ", type(a))
# float variable.
b=26.945
print("The type of variable having value", b, " is ", type(b))
# complex variable.
c=17+5j
print("The type of variable having value", c, " is ", type(c))
```

# 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.

# 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

# 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.

# Example

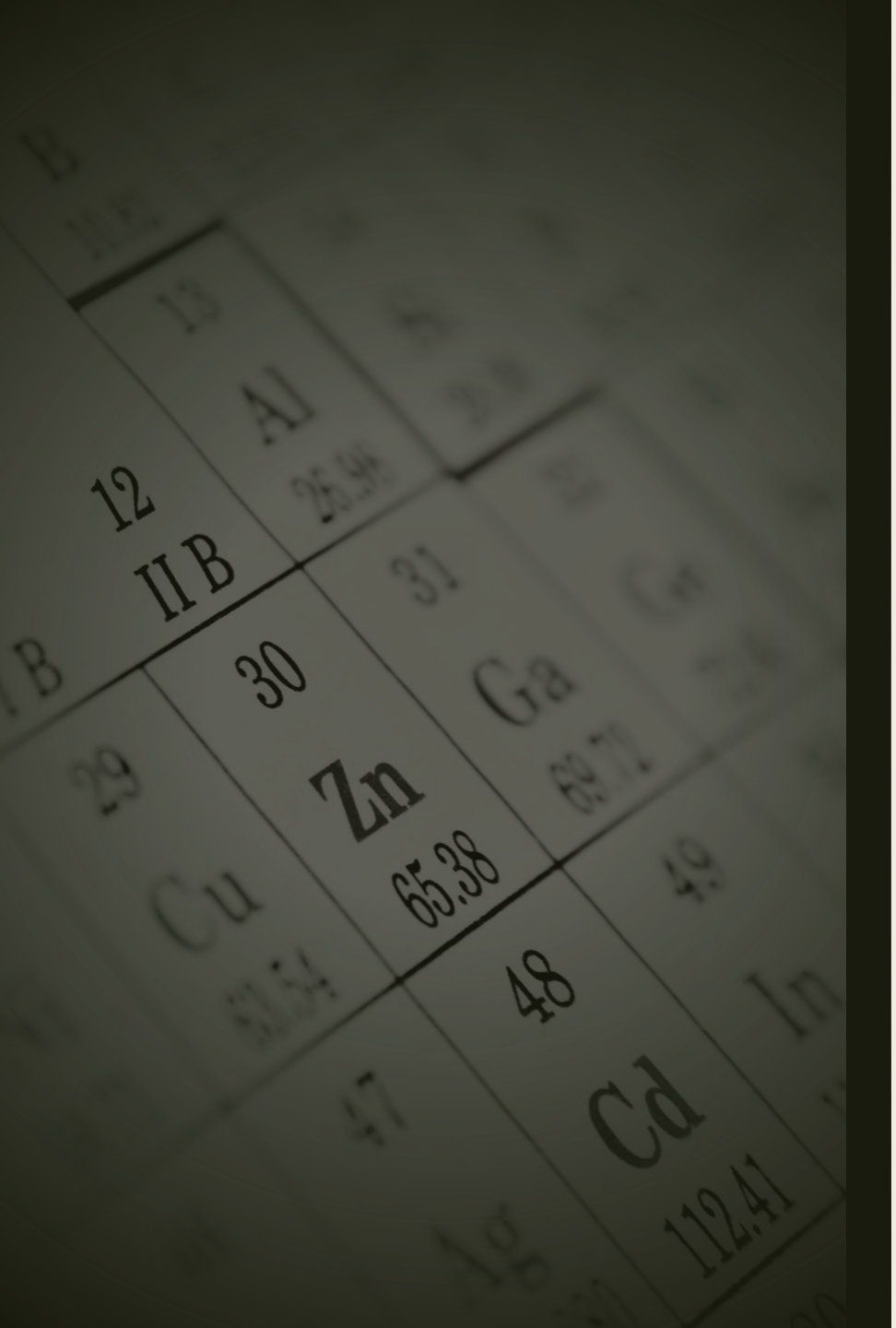
- l1 = [‘Rohan’, 345 , 5.23, ‘GG’, 80.2 ]
- l2 = [123, ‘GG’]
- print (l1) # Prints complete list
- print (l1[0]) # Prints first element of the list
- print (l1[1:3]) # Prints elements starting from 2nd till 3rd
- print (l1[2:]) # Prints elements starting from 3rd element
- print (l2 * 2) # Prints list two times
- print (l1 + l2) # Prints concatenated lists

- 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(mutable), while tuples are enclosed in parentheses ( ( ) ) and cannot be updated(immutable).
- Tuples can be thought of as **read-only** lists.

# Python Tuple Data Type

# Example

- t1 = ( ‘Rohan’, 345 , 5.23, ‘GG’, 80.2 )
- t2 = (123, ‘GG’)
- print (t1) # Prints the complete tuple
- print (t1[0]) # Prints first element of the tuple
- print (t1[1:3]) # Prints elements of the tuple starting from 2nd till 3rd
- print (t1[2:]) # Prints elements of the tuple starting from 3rd element
- print (t2 * 2) # Prints the contents of the tuple twice
- print (t1+ t) # Prints concatenated tuples




# 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)
```

- **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)

# Example

```
# loop to print number from 0 to 4
for i in range(5):
    print(i)
```

- 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 ([]).

# Python Dictionary

# Example

- d1 = {}
- d1['one'] = "This is one"
- d1[2] = "This is two"
- d2 = {'name': 'john','code':6734, 'dept': 'sales’}
- print (d1['one']) # Prints value for 'one' key
- print (d1[2]) # Prints value for 2 key
- print (d2) # Prints complete dictionary
- print (d2.keys()) # Prints all the keys
- print (d2.values()) # Prints all the values

# 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.

# Example

- a = True # display the value of a
- print(a) # display the data type of a
- print(type(a))

# PYTHON DATA TYPE CONVERSION

# 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)
```

# 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)

# 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)

# 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.

| Sr.No. | Function & Description |
|---|---|
| 1 | **int(x [,base])**<br>Converts x to an integer. base specifies the base if x is a string. |
| 2 | **long(x [,base] )**<br>Converts x to a long integer. base specifies the base if x is a string. |
| 3 | **float(x)**<br>Converts x to a floating-point number. |
| 4 | **complex(real [,imag])**<br>Creates a complex number. |
| 5 | **str(x)**<br>Converts object x to a string representation. |
| 6 | **repr(x)**<br>Converts object x to an expression string. |
| 7 | **eval(str)**<br>Evaluates a string and returns an object. |
| 8 | **tuple(s)**<br>Converts s to a tuple. |
| 9 | **list(s)**<br>Converts s to a list. |
| 10 | **set(s)**<br>Converts s to a set. |

| Sr.No. | Function & Description |
|---|---|
| 11 | **dict(d)**<br>Creates a dictionary. d must be a sequence of (key,value) tuples. |
| 12 | **frozenset(s)**<br>Converts s to a frozen set. |
| 13 | **chr(x)**<br>Converts an integer to a character. |
| 14 | **unichr(x)**<br>Converts an integer to a Unicode character. |
| 15 | **ord(x)**<br>Converts a single character to its integer value. |
| 16 | **hex(x)**<br>Converts an integer to a hexadecimal string. |
| 17 | **oct(x)**<br>Converts an integer to an octal string. |