Python- Variable Types

Lecture two

practical

Python- Variable Types

- Variables are nothing but reserved memory locations to store values.
- This means that when you create a variable you reserve some space in memory.
- Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.
- Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

Assigning values to variables

- Python declaration variables do not need explicit declaration to reserve memory space.
- The declaration happens automatically when you assign a value to a variable.
- The equal sign = is used to assign values to variables.
- The operand to the left of the = operator is the name of the variable and the operand to the right of of the = operator is the value stored in the variable.

Assigning values to variables

- For example –

```
counter= 100          # An integer assignment
miles=1000.0          # A floating point
name="John“           # A string
```

1. print(counter)
2. print(miles)
3. print(name)

- Here, 100, 1000.0 and "John" are the values assigned to counter, miles, and name variables, respectively. This produces the following result:

1. 100
2. 1000.0
3. John

Multiple Assignment

- Python allows you to assign a single value to several variables simultaneously. For example –
  - a = b = c = 1
- Here, an integer object is created with the value 1, and all three variables are assigned to the same memory location.
- You can also assign multiple objects to multiple variables. For example
  - a,b,c = 1,2,"john“
- Here, two integer objects with values 1 and 2 are assigned to variables a and b respectively, and one string object with the value "john" is assigned to the variable c.

Standard Data Types

- 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 standard data types that are used to define the operations possible on them and the method for each of them.
- Python has five standard data types:
  - Numbers
  - String
  - List
  - Tuple
  - Dictionary

Python Numbers

- Number data types store numeric values.
- Number objects are created when you assign a value to them.
- For example:
  - var1 = 1
  - var2 = 10
- You can also delete the reference to a number object by using the del statement.
- The syntax of the del statement is:
  - del var1
- You can delete a single object or multiple objects by using the del statement.
- For example:
  - del var1, var2

Python Numbers Example

- What is the output for the following code:

```
x=7
print(x)
del x
print(x)
```

- Python supports four different numerical types:
  - int (signed integers)

    X=7
  - long (long integers, they can also be represented in octal and hexadecimal)

    0xDEFABCECBDAECBFBAEl
  - float (floating point real values)

    15.20
  - complex (complex numbers)

    3.14j

Python Strings

- Strings in Python 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 working their way from -1 at the end.
- The plus + sign is the string concatenation operator and the asterisk * is the repetition operator.

Python Strings Example

- 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 Lists

- Lists are the most versatile of Python's compound data types.
- A list contains items separated by commas and enclosed within square brackets [ ].
- To some extent, lists are similar to arrays in C language. One difference between them is that all the items belonging to a list can be of different datatype.
- The values stored in a 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.

Python Lists Example

- For example:

```
list=['abcd',786,2.23,'john',70.2]
tinylist=[123,'john']
1. print(list)              # Prints complete list
2. print(list[0])           # Prints first element of the list
3. print(list[1:3])         # Prints elements starting from 2nd till 3rd
4. print(list[2:])          # Prints elements starting from 3rd element
5. print(tinylist*2)        # Prints list two times
6. print(list+tinylist)     # Prints concatenated lists
```

- This produce the following result:

```
1. ['abcd', 786, 2.23, 'john', 70.2]
2. abcd
3. [786, 2.23]
4. [2.23, 'john', 70.2]
5. [123, 'john', 123, 'john']
6. ['abcd', 786, 2.23, 'john', 70.2, 123, 'john']
```

Python Tuples

- A tuple is another sequence data type that is similar to the list.
- A 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:
  1. Lists are enclosed in brackets [ ] and their elements and size can be changed, while tuples are enclosed in parentheses ( ) and can not be updated.
  2. Tuples can be thought of as read-only lists.

Python Tuples Example

- For example:

```
tuple=('abcd',786,2.23,'john',70.2)
tinytuple=(123,'john')
1. print(tuple)              # Prints the complete tuple
2. print(tuple[0])           # Prints first element of the tuple
3. print(tuple[1:3])         # Prints elements of the tuple starting from 2nd till 3
4. print(tuple[2:])          # Prints elements of the tuple starting from 3rd element
5. print(tinytuple*2)        # Prints the contents of the tuple twice
6. print(tuple+tinytuple)    # Prints concatenated tuples
```

- This produce the following result:

1. ('abcd', 786, 2.23, 'john', 70.2)
2. abcd
3. (786, 2.23)
4. (2.23, 'john', 70.2)
5. (123, 'john', 123, 'john')
6. ('abcd', 786, 2.23, 'john', 70.2, 123, 'john')

Python Tuples Example

- 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]
- list[2]=1000 # Valid syntax with list
- print(list)
- tuple[2]=1000 # Invalid syntax with tuple

Python Dictionary

- Python's 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

- For example:

```
dict={}
dict['one']="This is one"
dict[2]="This is two“
tinydict={'name':'john','code':6734,'dept':'sales'}
1. print(dict['one'])          # Prints value for key 'one'
2. print(dict[2])              # Prints value for key 2
3. print(tinydict)             # Prints complete dictionary
4. print(tinydict.keys())      # Prints all the keys
5. print(tinydict.values())    # Prints all the values
```

- This produce the following result:

```
1. This is one
2. This is two
3. {'name': 'john', 'code': 6734, 'dept': 'sales'}
4. dict_keys(['name', 'code', 'dept'])
5. dict_values(['john', 6734, 'sales'])
```