PROGRAMMING WITH PYTHON

Karuna sheel

NIELIT kurukshetra

Python Variables

•Variable is a name which is used to refer memory location. Variable also known as identifier and used to hold value.

•In Python, we don't need to specify the type of variable because Python is a type infer language and smart enough to get variable type.

•Variable names can be a group of both letters and digits, but they have to begin with a letter or an underscore.

•It is recommended to use lowercase letters for variable name. Rahul and rahul both are two different variables.

Identifier Naming

•Variables are the example of identifiers. An Identifier is used to identify the literals used in the program. The rules to name an identifier are given below.

•The first character of the variable must be an alphabet or underscore (_).

•All the characters except the first character may be an alphabet of lower-case(a-z), upper-case (A-Z), underscore or digit (0-9).

•Identifier name must not contain any white-space, or special character (!, @, #, %, ^, &, *).

•Identifier name must not be similar to any keyword defined in the language.

•Identifier names are case sensitive for example my name, and MyName is not the same.

•Examples of valid identifiers : a123, _n, n_9, etc.

•Examples of invalid identifiers: 1a, n%4, n 9, etc.

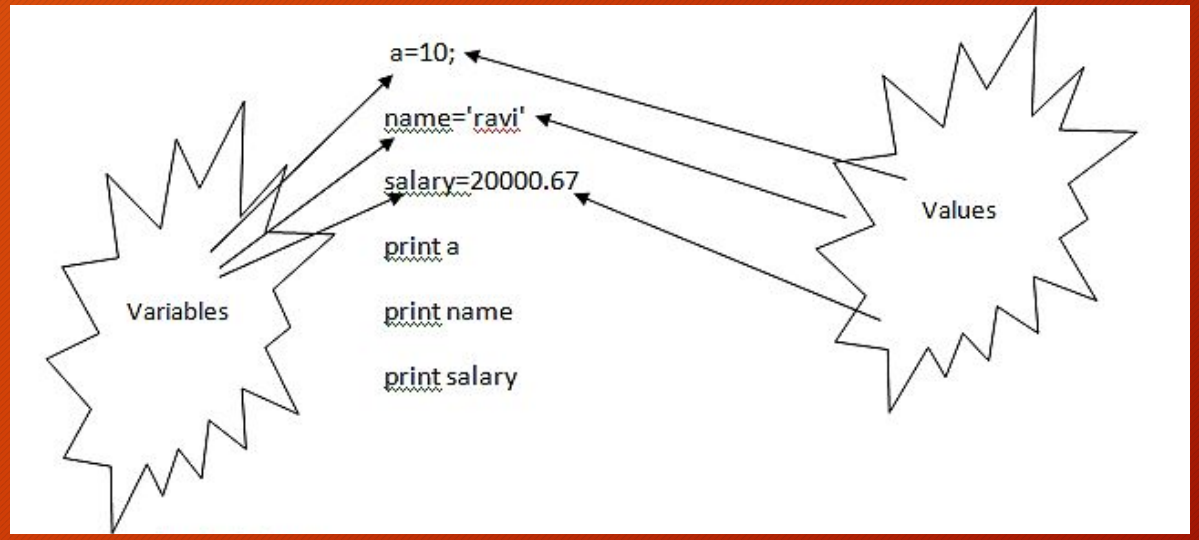
Declaring Variable and Assigning Values

Python does not bound us to declare variable before using in the application. It allows us to create variable at required time.

We don't need to declare explicitly variable in Python. When we assign any value to the variable that variable is declared automatically.

The equal (=) operator is used to assign value to a variable.

Eg:



Multiple Assignment

Python allows us to assign a value to multiple variables in a single statement which is also known as multiple assignment.

We can apply multiple assignments in two ways either by assigning a single value to multiple variables or assigning multiple values to multiple variables. Lets see given examples.

1. Assigning single value to multiple variables

Eg:

```
x=y=z=50
print x
print y
print z
```

Output:

```
>>>
50
50
50
>>>
```

2.Assigning multiple values to multiple variables:

Eg:

```
a,b,c=5,10,15
print a
print b
print c
```

Output:

```
>>>
5
10
15
>>>
```

The values will be assigned in the order in which variables appears.

Basic Fundamentals: Python Data Types

•Variables can hold values of different data types. Python is a dynamically typed language hence we need not define the type of the variable while declaring it. The interpreter implicitly binds the value with its type.

•Python enables us to check the type of the variable used in the program. Python provides us the type() function which returns the type of the variable passed.

•Consider the following example to define the values of different data types and checking its type.

```
a=10
b="Hi Python"
c = 10.5
print(type(a));
print(type(b));
print(type(c));
```

Output:

```
<type 'int'>
<type 'str'>
<type 'float'>
```

Standard data types

A variable can hold different types of values. For example, a person's name must be stored as a string whereas its id must be stored as an integer.

Python provides various standard data types that define the storage method on each of them. The data types defined in Python are:-

- Numbers
- String
- List
- Tuple
- Set
- Dictionary

Numbers

Number stores numeric values. Python creates Number objects when a number is assigned to a variable. For example;

a = 3 , b = 5 #a and b are number objects

Python supports 4 types of numeric data.

- int (signed integers like 10, 2, 29, etc.)
- long (long integers used for a higher range of values like 908090800L, -0x1929292L, etc.)
- float (float is used to store floating point numbers like 1.9, 9.902, 15.2, etc.)
- complex (complex numbers like 2.14j, 2.0 + 2.3j, etc.)

Python allows us to use a lower-case L to be used with long integers. However, we must always use an upper-case L to avoid confusion.

A complex number contains an ordered pair, i.e., x + iy where x and y denote the real and imaginary parts respectively).

String

- The string can be defined as the sequence of characters represented in the quotation marks. In python, we can use single, double, or triple quotes to define a string.
- String handling in python is a straightforward task since there are various inbuilt functions and operators provided.
- In the case of string handling, the operator + is used to concatenate two strings as the operation "hello"+" python" returns "hello python".
- The operator * is known as repetition operator as the operation "Python " *2 returns "Python Python ".

The following example illustrates the string handling in python.

```
str1 = 'hello there' #string str1
str2 = ' how are you' #string str2
print (str1[0:2]) #printing first two character using slice operator
print (str1[4]) #printing 4th character of the string
print (str1*2) #printing the string twice
print (str1 + str2) #printing the concatenation of str1 and str2
```

Output:

```
he
o
hello therehello there
hello there how are you
```

List

•Lists are similar to arrays in C. However; the list can contain data of different types. The items stored in the list are separated with a comma (,) and enclosed within square brackets [].

•We can use slice [:] operators to access the data of the list. The concatenation operator (+) and repetition operator (*) works with the list in the same way as they were working with the strings.

Consider the following example.

```
l  = [1, "hi", "python", 2]
print (l[3:]);
print (l[0:2]);
print (l);
print (l + l);
print (l * 3);
```

Output:

```
[2]
[1, 'hi']
[1, 'hi', 'python', 2]
[1, 'hi', 'python', 2, 1, 'hi', 'python', 2]
[1, 'hi', 'python', 2, 1, 'hi', 'python', 2, 1, 'hi', 'python', 2]
```

Syntax	Result	Description
Subjects [0]	Physics	This will give the index 0 value from the Subjects List.
Subjects [0:2]	Physics, Chemistry	This will give the index values from 0 till 2, but it won't include 2 the Subjects List.
Subjects [3] = 'Biology'	['Physics', 'Chemistry', 'Maths', 'Biology']	It will update the List and add 'Biology' at index 3 and remove 2.
del Subjects [2]	['Physics', 'Chemistry', 2]	This will delete the index value 2 from Subjects List.
len (Subjects)	['Physics, 'Chemistry', 'Maths', 2, 1, 2, 3]	This will return the length of the list
Subjects * 2	['Physics', 'Chemistry', 'Maths', 2] ['Physics', 'Chemistry', 'Maths', 2]	This will repeat the Subjects List twice.
Subjects [::-1]	[2, 'Maths', 'Chemistry', 'Physics']	This will reverse the Subjects List

Let's look at few operations that you can perform with Lists:

Tuple

•A tuple is similar to the list in many ways. Like lists, tuples also contain the collection of the items of different data types. The items of the tuple are separated with a comma (,) and enclosed in parentheses ().

•A tuple is a read-only data structure as we can't modify the size and value of the items of a tuple.

Let's see a simple example of the tuple.

```
t  = ("hi", "python", 2)
print (t[1:]);
print (t[0:1]);
print (t);
print (t + t);
print (t * 3);
print (type(t))
t[2] = "hi";
```

Output:

```
('python', 2)
('hi',)
('hi', 'python', 2)
('hi', 'python', 2, 'hi', 'python', 2)
('hi', 'python', 2, 'hi', 'python', 2, 'hi', 'python', 2)
<type 'tuple'>
Traceback (most recent call last):
  File "main.py", line 8, in <module>
    t[2] = "hi";
TypeError: 'tuple' object does not support item assignment
```

Dictionary

- Dictionary is an ordered set of a key-value pair of items. It is like an associative array or a hash table where each key stores a specific value. Key can hold any primitive data type whereas value is an arbitrary Python object.
- The items in the dictionary are separated with the comma and enclosed in the curly braces {}.

Consider the following example.

```
d = {1:'Jimmy', 2:'Alex', 3:'john', 4:'mike'};
print("1st name is "+d[1]);
print("2nd name is "+ d[4]);
print (d);
print (d.keys());
print (d.values());
```

Output:

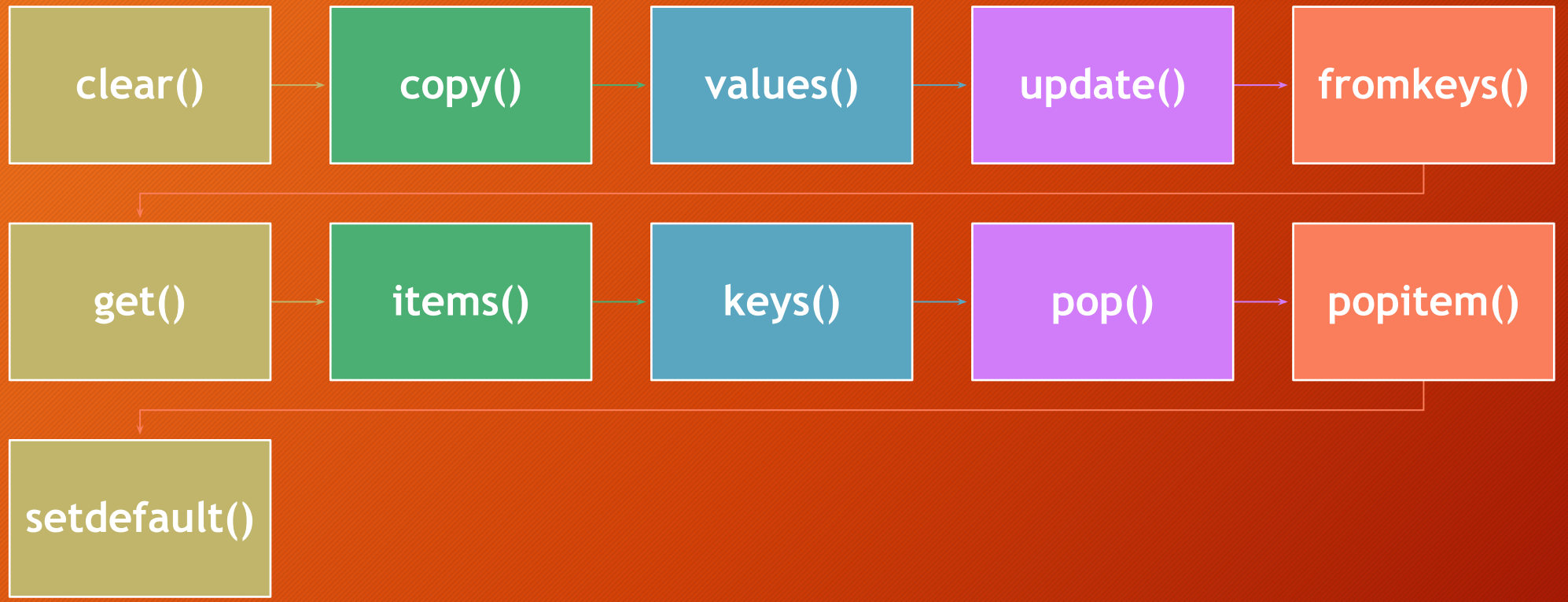
```
1st name is Jimmy
2nd name is mike
{1: 'Jimmy', 2: 'Alex', 3: 'john', 4: 'mike'}
[1, 2, 3, 4]
['Jimmy', 'Alex', 'john', 'mike']
```

More About Dictionary Data Type

Changing elements in a Dictionary:

```
Dict = {'Name' : 'Saurabh', 'Age' : 23}
Dict['Age'] = 32
Dict['Address'] = 'Starc Tower'
Output = {'Name' = 'Saurabh', 'Age' = 32, 'Address' = 'Starc Tower'}
```

Dictionary Methods:



Method	Description
clear()	Removes all items from the dictionary.
copy()	Returns a shallow copy of the dictionary.
fromkeys(seq[, v])	Returns a new dictionary with keys from seq and value equal to v (defaults to None).
get(key[,d])	Returns the value of the key. If the key does not exist, returns d (defaults to None).
items()	Return a new object of the dictionary's items in (key, value) format.
keys()	Returns a new object of the dictionary's keys.
pop(key[,d])	Removes the item with the key and returns its value or d if key is not found. If d is not provided and the key is not found, it raises KeyError.
popitem()	Removes and returns an arbitrary item (key, value). Raises KeyError if the dictionary is empty.
setdefault(key[,d])	Returns the corresponding value if the key is in the dictionary. If not, inserts the key with a value of d and returns d (defaults to None).
update([other])	Updates the dictionary with the key/value pairs from other, overwriting existing keys.
values()	Returns a new object of the dictionary's values

Dictionary Methods

```
car =     {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(car,"\n")

x = car.copy()
print(‘Using Copy Method’,x)

x = car.get("model") #get method
print('Using get method',x)

x = car.items()
print('Using items method',x)

x = ('key1', 'key2', 'key3')
y = 0
thisdict = dict.fromkeys(x, y)
print('using fromkeys method',thisdict)
```

```
car.pop("model")
print('Using pop method ',car)

car.popitem()
print('Using popitem method ',car)

x = car.setdefault("model", "Bronco")
print('using setdefault method ',x)

car.update({"color": "White"})
print('Using Update Method ',car)

car.clear()  #clear method is used to clear the items of
dictonaries
print('After Clear ',car)
```

Python Sets

A Set is an unordered collection of items. Every element is unique.

A Set is created by placing all the items (elements) inside curly braces {}, separated by a comma. Consider the example below:

Syntax

set_name = {1,2,3,4,5}

In Sets, every element has to be unique. Try printing the below code:

set_2 = {1,2,3,3,4,5}

Here 3 is repeated twice, but value will be print only once.

Let’s look at some Set operations:

Union:

Union of A and B is a set of all the elements from both sets. Union is performed using | operator. Consider the below example:

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}
print ( A | B)
```

Output = {1, 2, 3, 4, 5, 6}

Intersection:

Intersection of A and B is a set of elements that are common in both sets. Intersection is performed using & operator. Consider the example below:

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}
print ( A & B )
```

Output: {3,4}

Difference:

Difference of A and B (A – B) is a set of elements that are only in A but not in B. Similarly, B – A is a set of element in B but not in A. Consider the example below:

```
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
print(A - B)
```

Output: {1,2,3}