Variables in Python | Datatypes in Python

We use containers or boxes to store things used in day-to-day life. Similarly, even in Programming languages, we need some holders of information. These are called Variables. In the article, we will learn about Python Variables and their types. Let’s begin!

Variables in Python

Variables are the names that store the values. For example, if you want to store a number of students in a class then you can use it as a variable to store the count value.

Python does not have any need for declaring a variable. For example, the count of students in the above example is an integer value. In Python, you don’t need to write ‘int’ before the name ‘no\_of\_stds’.

You can create a variable and give it a value at that instant. You can use the token ‘=’, called the assignment operator. This will set the value on the right-hand side to the name on the left-hand side.

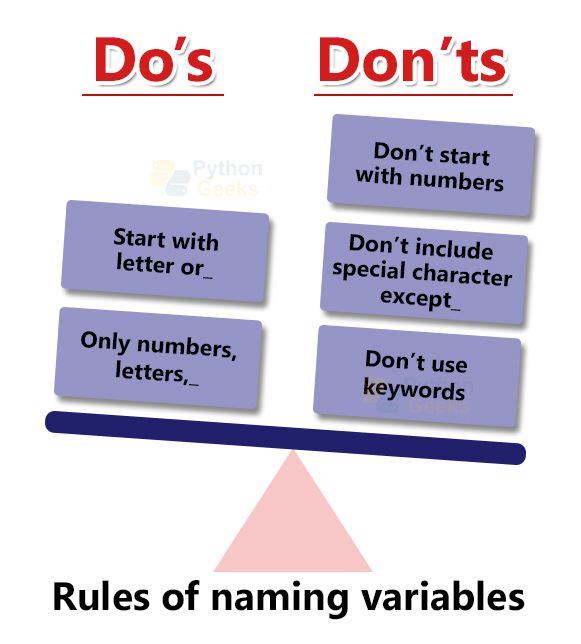
**Example of a Variable:**

no\_of\_stds = 40 # Assigning the value 40 to the variable 'no\_of\_stds'

In the above figure, the value 40 is assigned to the name ‘no\_of\_stds’.

Naming a Variable in Python

The name of the variable can be anything of the programmer’s choice, but it is desired to give a meaningful name and it should follow the below rules:

[](https://pythongeeks.org/wp-content/uploads/2021/07/Rules-of-naming-variables.jpg)

1. It must start with a letter or the underscore (\_).

**Example of Python variable names starting with letter or underscore:**

name="PythonGeeks"

\_no = 5

print(name)

print(\_no)

**Outputs:**

PythonGeeks  
5

2. It should not start with a number.

**Example on getting an error by starting name with number:**

4no = 5 # This gives an error because the variable name should not start with a number

**Output:**

SyntaxError: invalid syntax

3. It should only contain alpha-numeric characters and underscores (A-z, 0-9, and \_ ). It should not have any other characters or spaces.

**Example of Python Variable names with only alphanumeric characters and underscores:**

Name = "PythonGeeks"

number2=54

rate\_4=0.5

print(Name)

print(number2)

print(rate\_4)

**Outputs:**

PythonGeeks  
54  
0.5

**Example of getting an error by using a special character:**

rupees$=10

**Output:**

SyntaxError: invalid syntax

On running the above code, it showed an error because of the inclusion of the character ‘$’, which is not allowed.

4. The names are case-sensitive (i.e., Pythongeeks and PythonGeeks are treated as different variables).

**Example to show case sensitivity:**

number = 4

print(number)

print(Number)

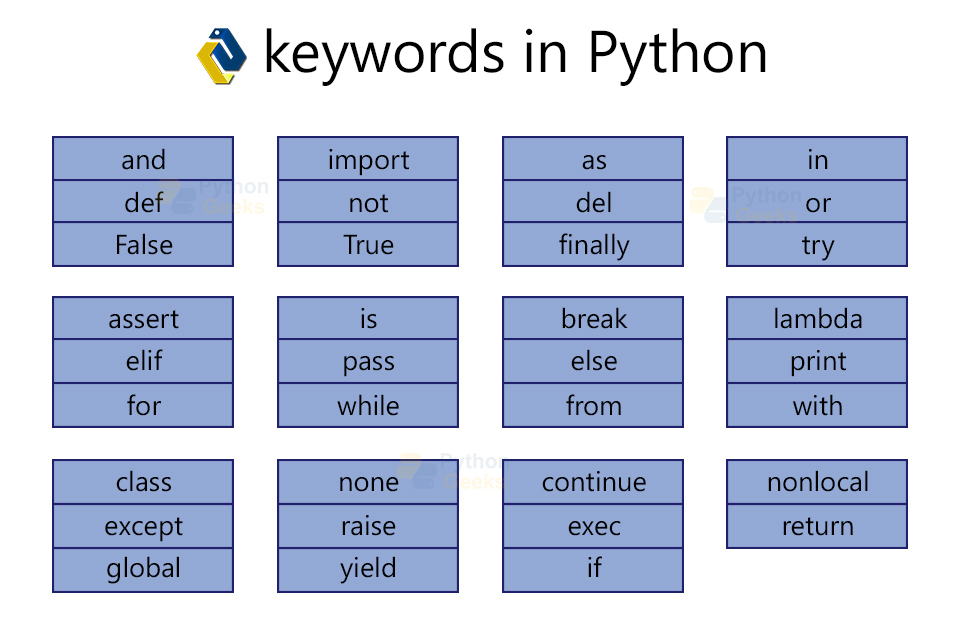
**Outputs:**

4

Traceback (most recent call last):  
File “<pyshell#2>”, line 1, in <module>  
print(Number)  
NameError: name ‘Number’ is not defined

The above code showed an error because of the case sensitivity of Python. It did not find the variable ‘Number’ and gave an error.

5. It should not be a keyword.

[](https://pythongeeks.org/wp-content/uploads/2021/07/keywords-in-Python.jpg)

**Example of getting an error on using a keyword as variable name:**

None=0

**Output:**

SyntaxError: cannot assign to None

Reassigning and Deleting the value of a Variable in Python

The value of a variable can be updated by reassigning the variable with the new value. For example, the count of the students in the previous example can be changed in the following way.

**Example of updating and deleting a Python variable:**

name="Python"

print("Name=",name)

name = "PythonGeeks" # Reassigning the value

print("New Name=",name)

del name #Deleting the variable

print("Deleted",name)

**Outputs:**

Name= Python

New Name= PythonGeeks

Traceback (most recent call last):  
File “<pyshell#5>”, line 1, in <module>  
print(“Deleted”,name)  
NameError: name ‘name’ is not defined

On changing the value, the new value is printed. After deleting the variable, when you try to print the value, it shows an error. This is because the variable will not be found.

Getting errors on wrong assigning:

1. We cannot use the variables before assigning them a value/creating it.

**Example of getting an error on accessing a value that does not exist:**

var

**Output:**

Traceback (most recent call last):  
File “<pyshell#0>”, line 1, in <module>  
var  
NameError: name ‘var’ is not defined

2. Always the variable name should be on the left side and the value should be on the right side of the assignment operator. Or else we will get a SyntaxError as shown below.

**Example of getting an error on the wrong assignment:**

'a'=name

**Output:**

SyntaxError: cannot assign to literal

More on Python Variables

1. Multiple assignments:

We can assign more than one value to the same number of variables, each separated by commas on either side.

**Example of multiple assignments:**

a,b,c=4,5,6

print(a,b,c)

**Output:**

4 5 6

You can also assign the same value to more than one variable using multiple assignments in the same line as shown below.

x=y=z=3.5

print(x,y,z)

**Output:**

3.5 3.5 3.5

2. Swapping values in Python

We can swap the values by using the above property of multiple assignments. For more understanding, look at the below example.

**Example of variable values swapping:**

a=5

b=19

a,b=b,a

print(a,b)

**Output:**

19 5

3. Deleting variables in Python:

We can delete variables using the del function. After deleting the variable, we cannot access them.  
**Example of deleting Python variable:**

var='abc'

print(var)

del var

print(var)

**Output:**

abc

Traceback (most recent call last):  
File “<pyshell#3>”, line 1, in <module>  
print(var)  
NameError: name ‘var’ is not defined

Types of Python Variables Based on Scope

Based on the scope of a variable, variables can be classified as either local or global variables.

a. Local Variables in Python

Local variables are the variables that are created in a function or a class and exist only while these constructs are executing. These variables cannot be accessed outside the function/ class.

**Example of Python local variable:**

def func():

var=3

print(var)

func()

print(var)

**Output:**

3

Traceback (most recent call last):  
File “<pyshell#5>”, line 1, in <module>  
print(var)  
NameError: name ‘var’ is not defined

b. Global variables in Python

These are the variables that are declared outside the functions/ classes. These will be available throughout the program.

**Example of Python global variable:**

var='a'

**def** func():

print(var)

func()

print(var)

**Output:**

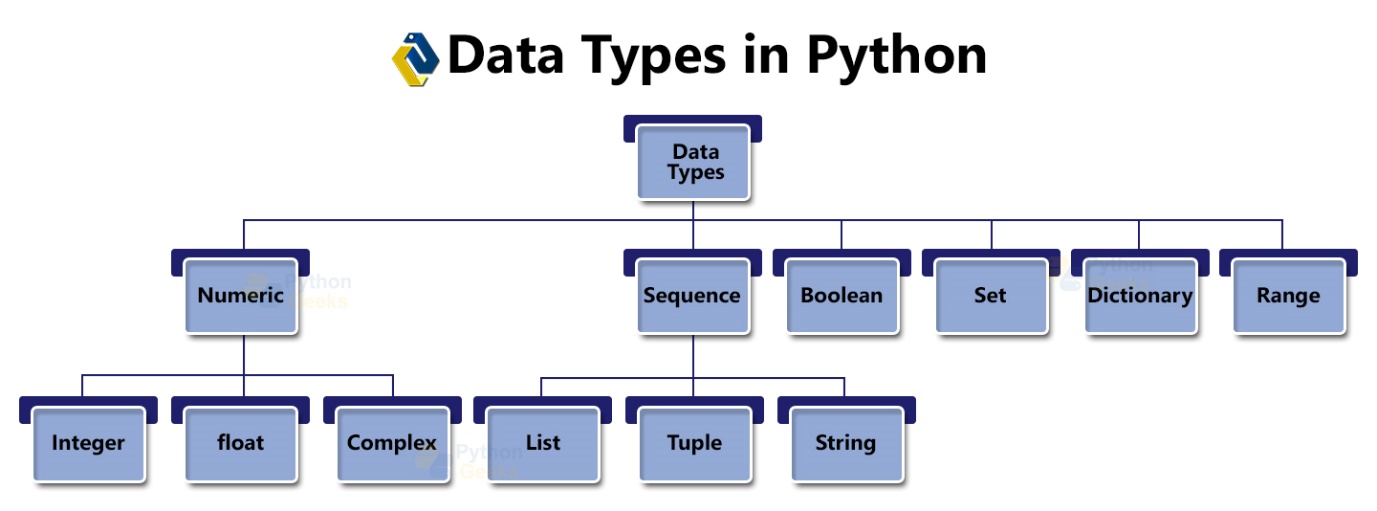
a

a

Data Types in Python

We come across different types of values in day-to-day life like count of objects which is an integer, weight in kg which is a decimal, name of a person which is a string, etc.

Similarly, different types of variables are used in Python to store different values. The built-in function ‘type ()’ can be used to find the data type of a variable. We can also use isinstance() to check if the value is of the specified data type or not. The data types include:

[](https://pythongeeks.org/wp-content/uploads/2021/07/Data-types-in-python.jpg)

Numeric Data Types in Python

**1. int:** This represents the integer values like 1,4, 99, etc.  
**2. float:** This is used to store the decimal values like 3.5, 4.0, etc.  
**3. complex:** This stores the complex value with the imaginary part represented with ‘j’, same as in math.  
**4. long:** This class was used to store integers of bigger length. But this does not exist in the Python versions 3.x.

**Example of creating numeric variables:**

count = 4 #int

percentage = 84.5 #float

a=4+3j #complex

print("Type of ",count," is ",type(count))

print("Type of ",percentage," is ",type(percentage))

print("Type of ",a," is ",isinstance(a,complex))

**Output:**

Type of 4 is <class ‘int’>  
Type of 84.5 is <class ‘float’>  
Type of (4+3j) is True

Sequence Data Types in Python

1. Lists in Python:

These are the data types that hold a collection of values. The elements are covered between square brackets. The list can be created in the following way:

**Example on creating a Python list:**

elements = [4,9,"PythonGeeks",50.4,**True**,'a'] #creating a list

print(elements)

**Output:**

[4, 9, ‘PythonGeeks’, 60, True, ‘a’]

**Operations that can be done on the list are:**

**a. Accessing elements of a list using an index:** We can access a value of a list by using the index, the location of the value. We can use the list name followed by the corresponding index in the square brackets. The index of the values of a list starts from 0.

**Example on accessing elements of a list using an index:**

print(elements[2]) #accessing the 3rd element of the list

**Output:**

PythonGeeks

**b. Slicing list to get a sub list:** We can slice a list to get multiple values of a list. We can give the indexing as ‘i:j’ to get the values from the index ‘i’th to the ‘j’th, excluding the ‘j’th one.

**Example on slicing list by indexing:**

print(elements[2:5]) # accessing elements from 3rd index to 5th index

print(elements[2:]) #accessing all elements from 3rd element

**Output:**

[‘PythonGeeks’, 50.4, True]  
[‘PythonGeeks’, 50.4, True, ‘a’]

**c. Finding the length of a list:** We can find the length of the list by using the built-in function len(). This takes the input as the list and returns the length as shown below.

**Example on finding the length of a list:**

print(len(elements)) #finding the length of the list

**Output:**

6

**d. Changing the elements of the list by reassigning the value:** We can reassign a value using the index by the following syntax:

list\_name[index]=new\_value

We can also add the values at the rightmost end of the list using the append() method and delete the values at the rightmost end using the pop() function.

**Example of changing a particular element of list:**

elements[3]=60 #changing the 4th element of the list

print(elements)

elements.append("Python") #adding element to the list

print(elements)

elements.pop() #deleting last element of the list and printing it

print(elements)

elements.insert(2,3.4) #inserting the value 3.4 at 2nd index

print(elements)

**Output:**

[4, 9, ‘PythonGeeks’, 60, True, ‘a’]  
[4, 9, ‘PythonGeeks’, 60, True, ‘a’, ‘Python’]  
‘Python’  
[4, 9, ‘PythonGeeks’, 60, True, ‘a’]  
[4, 9, 3.4, ‘PythonGeeks’, 60, True, ‘a’]

**e. Looping over a list in Python:**  
We can iterate over every element of a list using the loops. The below example shows an example of iterating using the ‘for’ loop. Here we used the iterator as the variable ‘i’ that goes through each element of the list till it reaches the end.

**Example of iterating over a Python list:**

list1=[1,2,3,4,5]

**for** i **in** list1:

print(i)

**Output:**

1  
2  
3  
4  
5

Other operations include:

|  |  |
| --- | --- |
| **Function** | **Operation** |
| clear() | removes all the elements from the list, but keeps the instance of the list |
| copy() | creates a copy of the list |
| extend() | add an element to the end of the list |
| count() | Finds the number of occurrences of the specified elements and returns it |
| index() | Finds the lowest location of the element |
| pop() | removes the element from the specified position or the last element,  if no argument is given |
| remove() | removes the specified element from the list |
| sort() | sorts the given list with homogeneous elements |
| reverse() | returns the list with elements in reverse order |

**Multidimensional lists:** Lists can be any dimension. An example of a 2D list is:

**Example of creating a multidimensional list:**

list\_2d=[[1,2,3],['a','b','c'],["Python","Geeks","PythonGeeks"]]

print(list\_2d)

**Output:**

[[1, 2, 3], [‘a’, ‘b’, ‘c’], [‘Python’, ‘Geeks’, ‘PythonGeeks’]]

2. Tuples in Python:

These are also a set of elements similar to the list. Elements are stored in between round brackets. The only difference is that the elements are immutable. If we try to change or delete an element we get an error.

**Example of creating a tuple and doing operations on it:**

tup=(1,2,"PythonGeeks",36.5,'a')

print(tup)

**Outputs:**

(1, 2, ‘PythonGeeks’, 36.5, ‘a’)

**Accessing the tuple values:**

The operations are the same as in lists except for those which change the values. We can access it using the index.

**Example of doing operations on it:**

tup=(1,2,"PythonGeeks",36.5,'a')

print(tup)

print(tup[2]) #accessing 3rd element

print(tup[1:4]) #accessing 2nd to 4th elements

print(tup[2:]) # accessing all the elements from 3rd one

**Outputs:**

(1, 2, ‘PythonGeeks’, 36.5, ‘a’)

PythonGeeks

(2, ‘PythonGeeks’, 36.5)

(‘PythonGeeks’, 36.5, ‘a’)

**Immutability of Python Tuples**

But since the tuples are immutable we get an error when we try to change the elements or delete any element.

**Example of doing operations on it:**

tup=(1,2,"PythonGeeks",36.5,'a')

tup[2]=4

del tup[3]

**Outputs:**

Traceback (most recent call last):  
File “<pyshell#6>”, line 1, in <module>  
tup[2]=4  
TypeError: ‘tuple’ object does not support item assignmentTraceback (most recent call last):  
File “<pyshell#7>”, line 1, in <module>  
del tup[3]  
TypeError: ‘tuple’ object doesn’t support item deletion

**Finding the length of Python tuple**

To find the length we use the built-in function len().

**Example of doing operations on it:**

tup=(1,2,"PythonGeeks",36.5,'a')

print(len(tup)) #finding length of tuple

**Outputs:**

5

**Other operations on Tuples in Python:**

We can also add two tuples or multiply the tuple by an integer. We can do this using the ‘+’ and ‘\*’ operators respectively.

**Example of doing addition and multiplication on tuples:**

tup1=(1,2,3,4,5)

tup2=(5,)

print("Addition:",tup1+tup2)

print("Multiplication:",tup2\*3)

**Output:**

Addition: (1, 2, 3, 4, 5, 5)

Multiplication: (5, 5, 5)

**Python Multidimensional Tuples:**

Like we have a multidimensional list, we have multidimensional tuples. These are the tuples with inner elements as tuples. These are also called nested tuples. The below example shows creating multi-dimensional tuple and accessing its elements:

**Example of Python multidimensional tuples:**

nest\_tup=((1,2,'a'),(5.4,6),((4,5),('r',9.0)))

print(nest\_tup[0])

print(nest\_tup[1][1])

print(nest\_tup[2][0][1])

**Output:**

(1, 2, ‘a’)

6  
5

3. Python String (str):

This data type handles the text like ‘a’, “PythonGeeks”, etc. These are surrounded by either single or double-quotes. We can also create multiline strings by enclosing them using three single or double quotes.  
**Example of creating String variables:**

string1='Python'

print("The type of ", string1,"is:",type(string1))

string2="PythonGeeks"

print("The type of ", string2,"is:",type(string2))

string3=''' Hello!

Welcome to PythonGeeks'''

print("The type of ", string3,"is:",type(string3))

**Output:**

The type of Python is: <class ‘str’>

The type of PythonGeeks is: <class ‘str’>

The type of Hello!  
Welcome to PythonGeeks is: <class ‘str’>

**Operations on Python Strings:**

**a. Accessing the characters of Python strings using indexes:** We can access the elements of the strings using indexes. We can use the string name followed by the corresponding index in the square brackets. In Python, the index of the strings starts from 0.

**Example of accessing a character of a string:**

name="PythonGeeks"

print(name[0]) #printing the 1st letter

**Output:**

P

In Python, indexing starts with 0. So, the character at 1st position will have an index of 0.

**b. Slicing Python string to get substring:** Slicing is to get more than one character of a string. We do indexing here too. To get the characters from the ‘i’th index to the ‘j’th, excluding the ‘j’th one, we can give the indexing as ‘i:j’.

**Example of slicing a string by using indexes:**

print(name[1:4])

**Output:**

yth

In the above example, the characters sliced are 1,2,3. We can see that the last index, which is 4, is not considered.

**c. Concatenation of Python strings:** We can use the ‘+’ operator to concatenate two or more strings. This appends the next string at the end of the previous string as shown below.

**Example of the concatenation of strings:**

print(name+name)

**Output:**

PythonGeeksPythonGeeks

**d. Immutability property of Python string:** This is the property of the string that does not allow us to change the characters of the string. Because of this, if we try modifying we get an error as shown below.

**Example to show immutable property of String:**

name[4]='t'

**Output:**

Traceback (most recent call last):  
File “<pyshell#5>”, line 1, in <module>  
name[4]=’t’  
TypeError: ‘str’ object does not support item assignment

**e. Python String formatting:**  
This is a method of combining the constant characters and values of the variables. We have three methods to do so. These are:

1) % operator  
2) f-string  
3) format() method

**Example on string formatting:**

name="abc"

print("His name is %s"%name)

print(f"His name is {name}")

print("His name is {}",format(name))

print("His name is {}".format(name))

print("His name is {0}".format(name))

**Output:**

His name is abc

His name is abc

His name is abc  
His name is abc

4. Python Boolean ( bool)

It contains only two values True and False.

**Example on creating a boolean variable:**

is\_Correct=**True**

is\_Wrong =**False**

print("Type of ",is\_Correct," is ",type(is\_Correct))

print("Type of ",is\_Wrong," is ",type(is\_Wrong))

**Output:**

Type of True is <class ‘bool’>  
Type of False is <class ‘bool’>

5. Python Sets

This is an unordered collection of unique elements. These use curly braces to hold the values. It can contain values of different data types.

**Example of creating a set and showing an error on indexing:**

set1={1,2,3,4}

print(set1)

set2={1,2,3,"PythonGeeks",45.6,'z'}

print(set2)

print(set1[0])

**Output:**

{1, 2, 3, 4}

{1, 2, 3, ‘z’, 45.6, ‘PythonGeeks’}

Operations on sets in Python:

**a. Accessing elements of a Python set**

We can access the whole set using the name of the set. But since it is unordered, it does not allow access using index. We get an error on doing so.

**Example of creating a set and showing an error on indexing:**

set1={1,2,3,4}

print(set1)

print(set1[0])

**Output:**

{1, 2, 3, 4}

Traceback (most recent call last):  
File “<pyshell#4>”, line 1, in <module>  
print(set1[0])  
TypeError: ‘set’ object is not subscriptable

**b. Adding an element to a set in Python:**

We can use the add() function to add elements to a list. For example,

**Example of adding elements to a set:**

set={1,2.4,'a','Python'}

set.add(3)

print(set)

**Output:**

{‘a’, 1, 2.4, ‘Python’, 3}

**c. Removing elements from a set:**

We use the remove() function to delete an element from a set. For example,

**Example of removing element from a set:**

set.remove('a')

print(set)

**Output:**

{1, 2.4, ‘Python’, 3}

**d. Python Membership:**

We can check if the element exists in a set using the membership operators, ‘in’ and ‘not in’. For example,

**Example of checking if element exists in a set:**

set={1,2,'t',5.6}

't' **in** set

3.4 not **in** set

5 **in** set

**Output:**

True

True

False

Other operations include:

|  |  |
| --- | --- |
| **Function** | **Operation** |
| clear() | Deletes all the items from the set and keeps the instance of the set |
| copy() | Creates a copy of the set and returns it |
| difference() | Finds the difference of the two sets and returns the resulting set |
| isdisjoint() | Finds if the sets have any common element and return True or False |
| issubset() | Finds if the set is a subset and returns True or False |
| symmetricdifference() | Finds the symmetric difference of two sets and  returns the resulting set |
| update() | Updates the two sets with the union |

6. Python Dictionaries

These are as a set of key-value pairs. This is similar to dictionaries used for knowing the meanings, where unique words/keys are mapped to meanings/values. The keys and values can be of any datatype.

**Example on creating Python dictionary:**

dict1={1:'a',2:'e',3:'i',4:'o',5:'u'}

print(dict1)

dict2={"Name":'abc',1:[1,2,3,4,5]}

print(dict2)

**Output:**

{1: ‘a’, 2: ‘e’, 3: ‘i’, 4: ‘o’, 5: ‘u’}

{‘Name’: ‘abc’, 1: [1, 2, 3, 4, 5]}

We can also create dictionaries from lists or tuples as shown below:

**Example on creating a dictionary:**

dict1=dict([['a','b'],[1,2]])

print(dict1)

dict2=dict([('a','b'),(1,2)])

print(dict2)

**Output:**

{‘a’: ‘b’, 1: 2}

{‘a’: ‘b’, 1: 2}

Operations that can be done on Python dictionaries:

**a. Accessing values using a key element:** The elements of a dictionary are not ordered, so we cannot index here. But we can use keys to access corresponding values. We can also access it using the get() method.

**Example on accessing a value using key:**

dic={1:'a',2:'b',3:'c',4:'D',"PythonGeeks":5}

print(dic['PythonGeeks']) #accessing using key

print(dic.get(1))

**Output:**

5

a

**b. Reassigning values and deleting elements in Python:** We can reassign a value corresponding to a key by giving the key as the argument.

The syntax is:

dict\_name[key]=new\_value

Similarly, we can delete using the pop() function by giving the corresponding key as the argument.

**Example of changing the value of a key element :**

dic['PythonGeeks']=6

print(dic)

dic.pop(4)

dic

**Output:**

{1: ‘a’, 2: ‘b’, 3: ‘c’, 4: ‘D’, ‘PythonGeeks’: 6}

‘D’  
{1: ‘a’, 2: ‘b’, 3: ‘c’, ‘PythonGeeks’: 6}

**c. Getting all the keys and values as a list:** We can get all the keys and the values using the keys() and values() functions respectively.

**Example on getting the keys and values as lists:**

print(dic.keys()) #accessing all the keys as a list

print(dic.values()) #accessing all the values

**Output:**

dict\_keys([1, 2, 3, 4, ‘PythonGeeks’])

dict\_values([‘a’, ‘b’, ‘c’, ‘D’, 6])

Other operations include:

|  |  |
| --- | --- |
| **Function** | **Operation** |
| copy() | Creates a copy of the dictionary and returns it |
| clear() | Deletes all the elements of the dictionary and keeps the instance |
| items() | Returns all the key-value pairs as tuples collected in the form of a list |
| keys() | Returns a list of all the keys |
| update() | Updates the dictionary with all the key-value pairs |
| values() | Returns a list of all the values |
| setdefault() | Returns the value of a specified key. If the key does not exist, then inserts the key with the specified default value and returns it. |

7. Python Range

This is a data type that creates a collection of values in the specified interval. This is also a method that is used usually in loops.

**Example on creating a range:**

var=range(1,10) #creating a range of values from 1 to 9

print(var)

**for** i **in** var:

print(i)

**Output:**

range(1, 10)

1  
2  
3  
4  
5  
6  
7  
8  
9

Conversion of One Data Type to Another in Python

On understanding different data types, let us see how to convert from one to another.

Conversion can be done from one data type to another data type if it is an appropriate one. We can typecast to do so. The data type is preceded by the value in the braces. The most common ones are int(), float(), complex(), bool(), str(), list(), set(), dict(), tuple().

**1. int() converts the value given as an argument to the data type int**

**Example on conversion to int data type :**

a=int(3.41) #typecasting to integer

print("The value in a is ",a,"and its type is",type(a))

**Output:**

The value in a is 3 and its type is <class ‘int’>

**2. float() converts the value into float type**

**Example on conversion to float data type :**

b=float(40)#typecasting to float

print("The value in b is ",b,"and its type is",type(b))

**Output:**

The value in b is 40.0 and its type is <class ‘float’>

**3. complex () converts the value into complex data type**

**Example on conversion to complex data type :**

c=complex(7.8)

print("The value in c is ",c,"and its type is",type(c))

**Output:**

The value in c is (7.8+0j) and its type is <class ‘complex’>

**4. bool() returns either boolean False if the value is 0/empty else boolean True**

**Example on conversion to boolean data type :**

boolean=bool(5)

print("The value in boolean is ",boolean,"and its type is",type(boolean))

boolean2=bool("")

print("The value in boolean2 is ",boolean2,"and its type is",type(boolean2))

**Output:**

The value in boolean is True and its type is <class ‘bool’>

The value in boolean2 is False and its type is <class ‘bool’>

**5. str() converts the input into a string value.**

**Example on conversion to string data type :**

d=str(10.5)#typecasting to String

print("The value in d is ",d,"and its type is",type(d))

**Output:**

The value in d is 10.5 and its type is <class ‘str’>

**6. list() converts the iterable given into a list**

**Example on conversion to list data type :**

list1=list("abcd")

print("The value in list1 is ",list1,"and its type is",type(list1))

**Output:**

The value in list1 is [‘a’, ‘b’, ‘c’, ‘d’] and its type is <class ‘list’>

**7. set() converts the iterable into a set type**

**Example on conversion to set data type :**

set1=set([1,2,3,1])

print("The value in set1 is ",set1,"and its type is",type(set1))

**Output:**

The value in set1 is {1, 2, 3} and its type is <class ‘set’>

**8. dict() maps the values given as arguments and returns the corresponding dictionary.**

**Example on conversion to dictionary data type :**

dict1=dict(name='abc',age=17, gender='Female')

print("The value in dict1 is ",dict1,"and its type is",type(dict1))

**Output:**

The value in dict1 is {‘name’: ‘abc’, ‘age’: 17, ‘gender’: ‘Female’} and its type is <class ‘dict’>

**9. tuple() converts the iterable into a tuple type**

**Example on conversion to tuple data type :**

tup1=tuple("Python")

print("The value in tup1 is ",tup1,"and its type is",type(tup1))

**Output:**

The value in tup1 is (‘P’, ‘y’, ‘t’, ‘h’, ‘o’, ‘n’) and its type is <class ‘tuple’>

Illegal conversion from one data type to another:

However, illegal conversion cannot be done like converting a word/character to an integer. And converting an integer to a list is also not possible because integers are not iterable. These will give an error on execution as shown below.

**Example of illegal typecasting:**

n=int('PythonGeeks')

list1=list(4)

**Output:**

Traceback (most recent call last):  
File “<pyshell#0>”, line 1, in <module>  
n=int(‘PythonGeeks’)  
ValueError: invalid literal for int() with base 10: ‘PythonGeeks’

Traceback (most recent call last):  
File “<pyshell#1>”, line 1, in <module>  
list1=list(4)  
TypeError: ‘int’ object is not iterable

Other Python conversions :

1. repr(): Converts the input into a string expression  
2. frozenset(): Converts the input into a frozenset  
3. chr(): Converts an integer into a character according to ASCII values  
4. unichr(): Converts an integer into a Unicode character  
5. ord(): Converts a single character into its integer value according to ASCII values  
6. hex(): Converts an integer into hexadecimal form and returns it as a string  
7. oct(): Converts an integer into octal form and returns it as a string  
8. eval(): Evaluates a string argument and returns an object

Implicit Conversions of Data types in Python

Till now what we have discussed are the explicit conversions, where we used the built-in functions to convert from one data type to another. Besides these, there are cases where we see automatic conversions. These are called implicit conversions.

For example, if we add 3.4 and 5, we get that output as 8.4. We see that integer 5 gets converted to float. This example is shown below.

**Example of implicit type casting:**

a=3.4

b=5

c=a+b

print("The type of ",a," is:",type(a))

print("The type of ",b," is:",type(b))

print("The type of ",c," is:",type(c))

**Output:**

The type of 3.4 is: <class ‘float’>

The type of 5 is: <class ‘int’>

The type of 8.4 is: <class ‘float’>

Interview Questions on variables in Python

Learning a lot of concepts and practicing coding, let us discuss some questions on the above topics.

Q1. Can any data type be typecast to boolean? Show using some coding examples.

Ans**.**Yes. Any data type can be cast to boolean type, including the sequence data types. The value will be “False” if it is empty or 0 else “True”. Below are some examples.

**Examples of type casting to boolean:**

bool(0)

bool(90)

bool(144.5)

bool([1,'a',"PythonGeeks",9.5])

bool("")

bool("PythonGeeks")

**Output:**

False

True

True

True

False

True

Q2. Write the code to store the String Python’sString in a variable and print its value.

Ans**.** Since the single quote (‘) is used at both ends of a string, an escape sequence is needed to store the quotes. The following code represents it.

**Example of storing a string with a single quote:**

string = "Python\'sString"

print(string)

**Output:**

Python’sString

Q3. What is the data type of the sequences obtained by slicing? Show it using a coding example.

Ans. The sequence type data types are list, tuple, etc. On slicing, we get a subcollection of elements. The data type of the result obtained from slicing is the same datatype as that of the collection.

**Example on slicing a tuple and finding the type():**

tup=(1,'a',"PythonGeeks",9.5)

type(tup)

type(tup[1:3])

**Output:**

<class ‘tuple’>

<class ‘tuple’>

**Example on slicing a list and finding the type():**

list1=[1,'a',"PythonGeeks",9.5]

type(list1)

type(list1[1:3])

**Output:**

<class ‘list’>

<class ‘list’>

Q4. In Python 3, what is the type of range of values? Show it by creating a range of values and check the data type.

Ans. In Python 3, the range of values comes under the class of ‘range’. This is shown in the below example

**Example on checking the type of range of values:**

values=range(1,5) #creating a range of values from 1 to 4

print(type(values)) #checking its type

**Output:**

<class ‘range’>

Q5. Write a program to print the last element of the list.

Ans. First, we can find the length of the list using the ‘len()’ function. And since the indexes start from 0, the last element will be at the index of length -1.

**Example of finding the last element of the list:**

list1=[1.6,'c',5,"PythonGeeks"]

length=len(list1)

print(list1[length-1])

**Output:**

PythonGeeks