Session- 3

06 July 2024 09:15 PM

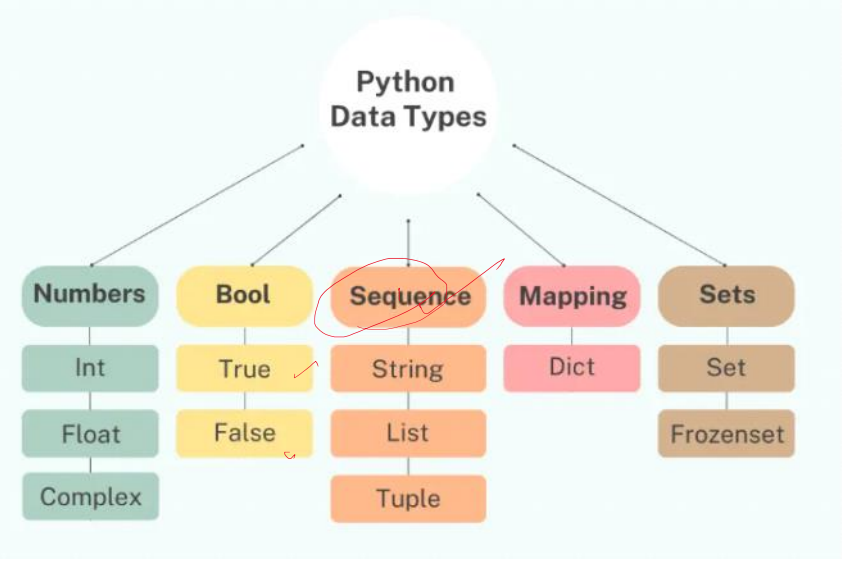
By Alok Ranjan
Linkedin- www.linkedin.com/in/alok-ranjandigu-coder
GitHub- https://github.com/alokyadav2020

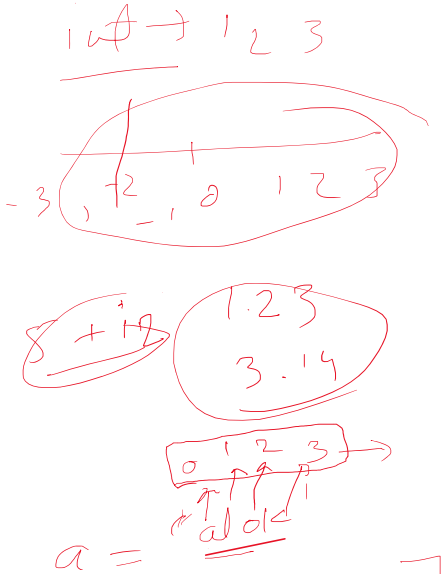
Agenda

- Data structures in Python
- Operators in Python
- Control statement - Conditions

DATA TYPES

Every value belongs to a specific data type in Python. Data type identifies the type of data values a variable can hold and the operations that can be performed on that data.

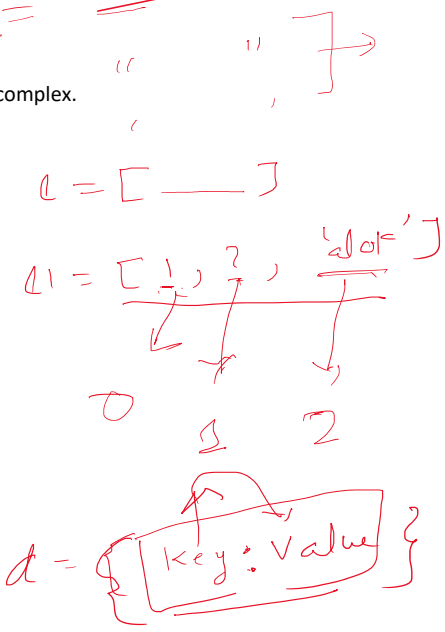




Number

Number data type stores numerical values only. It is further classified into three different types: int, float and complex.

Type/ Class	Description	Examples
int	integer numbers	–12, –3, 0, 125, 2
float	real or floating point numbers	–2.04, 4.0, 14.23
complex	complex numbers	3 + 4j, 2 – 2j



Sequence

1. **String**

Sequence

1. **String**

String is a group of characters. These characters may be alphabets, digits or special characters including spaces.
We cannot perform numerical operations on strings, even when the string contains a numeric value, as in str2

2. **List**

List is a sequence of items separated by commas and the items are enclosed in square brackets [ ].

3. **Tuple**

Tuple is a sequence of items separated by commas and items are enclosed in parenthesis ( ). This is unlike list, where values are enclosed in brackets [ ]. Once created, we cannot change the tuple.

Set

Set is an unordered collection of items separated by commas and the items are enclosed in curly brackets { }. A set is similar to list, except that it cannot have duplicate entries. Once created, elements of a set cannot be changed.
duplicate elements are not included in set.

None

None is a special data type with a single value. It is used to signify the absence of value in a situation. None supports no special operations, and it is neither same as False nor 0 (zero).

Ex. >>> myVar = None
>>> print(type(myVar))
>>> print(myVar) None 5.7.5

Mapping

Mapping is an unordered data type in Python. Currently, there is only one standard mapping data type in Python called dictionary. (A) Dictionary Dictionary in Python holds data items in key-value pairs. Items in a dictionary are enclosed in curly brackets { }. Dictionaries permit faster access to data. Every key is separated from its value using a colon (:) sign. The key : value pairs of a dictionary can be accessed using the key. The keys are usually strings and their values can be any data type. In order to access any value in the dictionary, we have to specify its key in square brackets [ ].

Ex. create a dictionary
>>> dict1 = {'Fruit':'Apple', 'Climate':'Cold', 'Price(kg)':120}
>>> print(dict1) {'Fruit': 'Apple', 'Climate': 'Cold', 'Price(kg)': 120}
>>> print(dict1['Price(kg)']) 120

OPERATORS

An operator is used to perform specific mathematical or logical operation on values. The values that the operators work on are called operands. For example, in the expression 10 + num, the value 10, and the variable num are operands and the + (plus) sign is an operator.

Python divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

Python Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Python Assignment Operators

Assignment operators are used to assign values to variables:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3
:=	print(x := 3)	x = 3 print(x)

Python Comparison Operators

Comparison operators are used to compare two values:

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y

Python Comparison Operators

Comparison operators are used to compare two values:

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Python Logical Operators

Logical operators are used to combine conditional statements:

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Python Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Python Membership Operators

Membership operators are used to test if a sequence is presented in an object:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

Operator	Name	Description	Example
&	AND	Sets each bit to 1 if both bits are 1	x & y
|	OR	Sets each bit to 1 if one of two bits is 1	x | y
^	XOR	Sets each bit to 1 if only one of two bits is 1	x ^ y
~	NOT	Inverts all the bits	~x
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off	x << 2
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off	x >> 2

Python If ... Else

Python Conditions and If statements

Python supports the usual logical conditions from mathematics:

- Equals: `a == b`
- Not Equals: `a != b`
- Less than: `a < b`
- Less than or equal to: `a <= b`
- Greater than: `a > b`
- Greater than or equal to: `a >= b`

These conditions can be used in several ways, most commonly in "if statements" and loops.
An "if statement" is written by using the `if` keyword.

Ex.

```
a = 33
b = 200
if b > a:
 print("b is greater than a")
```

In this example we use two variables, a and b, which are used as part of the if statement to test whether b is greater than a. As a is 33, and b is 200, we know that 200 is greater than 33, and so we print to screen that "b is greater than a".

Indentation

Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly-brackets for this purpose.

Ex.

```
a = 33
b = 200
if b > a:
print("b is greater than a")# you will get an error
```

Elif

The `elif` keyword is Python's way of saying "if the previous conditions were not true, then try this condition".

Ex.

```
a = 33
b = 33
if b > a:
 print("b is greater than a")
elif a == b:
 print("a and b are equal")
```

In this example a is equal to b, so the first condition is not true, but the elif condition is true, so we print to screen that "a and b are equal".

Else

The else keyword catches anything which isn't caught by the preceding conditions.

Ex.

```
a = 200
b = 33
if b > a:
 print("b is greater than a")
elif a == b:
 print("a and b are equal")
else:
 print("a is greater than b")
```

In this example a is greater than b, so the first condition is not true, also the elif condition is not true, so we go to the else condition and print to screen that "a is greater than b".
You can also have an else without the elif:

Ex.

```
a = 200
b = 33
if b > a:
 print("b is greater than a")
else:
 print("b is not greater than a")
```

Short Hand If

If you have only one statement to execute, you can put it on the same line as the if statement.

Example

One line if statement:

```
if a > b: print("a is greater than b")
```

Short Hand If ... Else

If you have only one statement to execute, one for if, and one for else, you can put it all on the same line:

Example

One line if else statement:

```
a = 2

b = 330

print("A") if a > b else print("B")
```

This technique is known as **Ternary Operators**, or **Conditional Expressions**.

Example

One line if else statement, with 3 conditions:

```
a = 330

b = 330

print("A") if a > b else print("=") if a == b else print("B")
```

And

The `and` keyword is a logical operator, and is used to combine conditional statements:

Example

Test if `a` is greater than `b`, AND if `c` is greater than `a`:

```
a = 200
b = 33
c = 500
if a > b and c > a:
  print("Both conditions are True")
```

Or

The `or` keyword is a logical operator, and is used to combine conditional statements:

Example

Test if `a` is greater than `b`, OR if `a` is greater than `c`:

```
a = 200
b = 33
c = 500
if a > b or a > c:
  print("At least one of the conditions is True")
```

Not

The `not` keyword is a logical operator, and is used to reverse the result of the conditional statement:

Example

Test if `a` is NOT greater than `b`:

```
a = 33
b = 200
if not a > b:
  print("a is NOT greater than b")
```

Nested If

You can have `if` statements inside `if` statements, this is called *nested* `if` statements.

Example

```
x = 41

if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

The pass Statement

`if` statements cannot be empty, but if you for some reason have an `if` statement with no content, put in the `pass` statement to avoid getting an error.

Example

```
a = 33
```

```
b = 200

if b > a:
  pass
```