Chapter-4: Python Language Fundamentals

Topics:

Character set of python

Tokens

1. Keywords
2. Identifiers
3. Datatypes
4. Operators

Literals

Character Set of Python

The set of characters supported or used in Python is called a **character set**.

There are **two types of character sets** in Python:

1. ASCII
2. Unicode

ASCII

- ASCII stands for **American Standard Code for Information Interchange**.
- ASCII supports **256 characters (0–255)**.
- It supports:
  - English alphabets (A–Z, a–z)
  - Digits (0–9)

    - Special characters (+, -, <, >, ?, /, ;, :, ", etc.)

Unicode

- Unicode is a **superset of ASCII**.
- Unicode supports characters from **English and other languages** such as Hindi, Tamil, Telugu, etc.
- It also supports **emojis and special characters**.
- Unicode supports **297,334 assigned characters**.

Example (ASCII variables)

```
>>> a = 10
>>> b = 20
>>> a
10
>>> b
20
```

Example (Unicode variables – Hindi)

```
>>> अ = 10
>>> आ = 20
>>> अ + आ
30
```

Example (Unicode variables – Telugu)

```
>>> క = 10
>>> ఖ = 20
>>> క + ఖ
30
```

Printing Unicode Text

```
print("పైఢాన్‌కు స్వాగతం 🙂")
print("पायथन में आपका स्वागत है 🙂")
```

Output:

```
పైఢాన్‌కు స్వాగతం 🙂
पायथन में आपका स्वागत है 🙂
```

Tokens

A **token** is the smallest individual unit within a program.

Types of Tokens:

1. Keywords
2. Identifiers
3. Literals
4. Datatypes
5. Operators

Keywords

Keywords are **language-defined words**.
Each keyword has a **specific meaning** and is used to perform a **specific operation or task** in Python.

How to find the keyword list in Python?

```
>>> import keyword
>>> keyword.kwlist
```

Output:

```
['False', 'None', 'True', 'and', 'as', 'assert',
'async', 'await','break', 'class', 'continue', 'def',
'del', 'elif', 'else','except', 'finally', 'for',
'from', 'global', 'if', 'import','in', 'is','lambda',
'nonlocal', 'not', 'or', 'pass', 'raise',
 'return', 'try', 'while', 'with', 'yield']
```

Python 3.14 Keywords

- Python 3.14 supports **35 keywords**

Soft Keywords in Python 3.14

```
>>> keyword.softkwlist
```

Output:

```
['_', 'case', 'match', 'type']
```

- Python 3.14 supports **4 soft keywords**

Total Keywords in Python

Total number of keywords supported by Python = 39

Identifiers

An **identifier** is a **user-defined word** or **programmer-defined word**.
Identifiers are used to identify programming elements such as:

1. Variable name
2. Constant name
3. Function name
4. Module name
5. Class name
6. Package name

Rules for Creating Identifiers

An identifier is a **single word** created using:

1. Alphabets **A–Z** or **a–z** (uppercase or lowercase)
2. Digits **0–9**
3. Underscore **_** (special character)

Rule 1: Keywords cannot be used as identifiers

```
>>> rollno = 123
>>> rollno
123
```

```
>>> pass = 100
SyntaxError: invalid syntax

>>> def = 1
SyntaxError: invalid syntax

>>> break = 15
SyntaxError: invalid syntax

>>> PASS = 100
>>> PASS
100
```

Note: Python keywords cannot be used as identifiers, but uppercase words like PASS are allowed because Python is **case-sensitive**.

Rule 2: Identifier should not start with a digit

```
>>> number1 = 25

>>> 2number = 50
SyntaxError: invalid decimal literal

>>> _number = 50
>>> _number
50

>>> 3num = 100
SyntaxError: invalid decimal literal
```

Rule 3: Identifier allows only one special character (_)

Only the **underscore (_)** is allowed as a special character in identifiers.

```
>>> amt$ = 100
SyntaxError: invalid syntax

>>> $amt = 5
SyntaxError: invalid syntax

>>> _amt = 5
>>> amt_ = 5
>>> _ = 10

>>> _amt
5
>>> amt_
5
>>> _
10

>>> student_rollno = 1
>>> student_rollno
1

>>> a_b_c = 100
>>> a_b_c
100
```

Note:

- $, @, #, etc. are **not allowed**
- Only _ (underscore) is allowed

- Underscore can be used **at the beginning, middle, or end**

Rule 4: Maximum length of identifier is unlimited

Python allows identifiers of **any length**.

```
>>> aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa = 100
>>> aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
100

>>> bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb = 1
```

Rule 5: Python is case-sensitive

Python treats **uppercase and lowercase letters as different**.

```
>>> a = 100
>>> A = 200

>>> a
100
>>> A
200
```

Note:
a and A are considered **different identifiers** in Python.

Rule 6: There should not be any space within an identifier

Identifiers **must not contain spaces**.

```
>>> student rollno number = 1
SyntaxError: invalid syntax

>>> employee salary = 50000
SyntaxError: invalid syntax
```

Correct way:

```
student_rollno_number = 1
employee_salary = 50000
```

Naming Conventions (PEP 8)

PEP stands for **Python Enhancement Proposal**.
It is a document that defines **enhancements and coding standards** for the Python language.
PEP 8 defines **naming conventions**.

---

1. Variable Naming Convention

- Variable names should be in **lowercase**.
- If a variable name has multiple words, separate them using **underscore (_)**.

Examples:

```
rollno
customer_balance
user_name
```

---

2. Function Naming Convention

- Function names should be in **lowercase**.
- If a function name has multiple words, separate them using **underscore (_)**.

Examples:

```
deposit()
withdraw()
login_user()
```

---

3. Class Naming Convention

- Class names should start with a **capital letter**.
- If a class name has multiple words, use **CapitalizedWords (CamelCase)**.

Examples:

```
Account
Customer
SalariedEmployee
```

---

4. Constant Naming Convention

- Constant names should be written in **capital letters**.
- If a constant name has multiple words, separate them using **underscore (_)**.

Examples:

```
PI
MAX_BALANCE
MIN_BALANCE
```

---

5. Module Naming Convention

- Module names should be in **lowercase**.

Examples:

```
chess
numpy
pandas
```

Data Types and Literals

A **literal** is a value or data that **never changes**.

- Literals represent values stored in **main memory (RAM)** using data types.
- Data types are used to **reserve space for data**.
- Data is never processed without being stored in **memory (RAM)**.

Classification of Python Data Types

Python data types are classified into **two categories**:

1. Scalar data types

    a. Int
    b. Float
    c. Complex
    d. Bool
    e. NoneType

2. Collection Data types

    a. Sequence types
        i. List
        ii. Tuple
        iii. Range
        iv. Str
        v. Bytes
        vi. Bytearray
    b. Sets
        i. Set
        ii. Frozenset
    c. Mapping
        i. dict

Python support 14 standard data types (5+9)

Literals :

Literals are nothing but values. Python support different types of literals
1. Numeric Literals
2. Non Numeric Literals

1.Numeric Literals
Numeric literals are nothing but numbers.
Python supports various types of numeric values.

1. Integer literal

2. Float literal

3. Complex literal

Integer literal or value

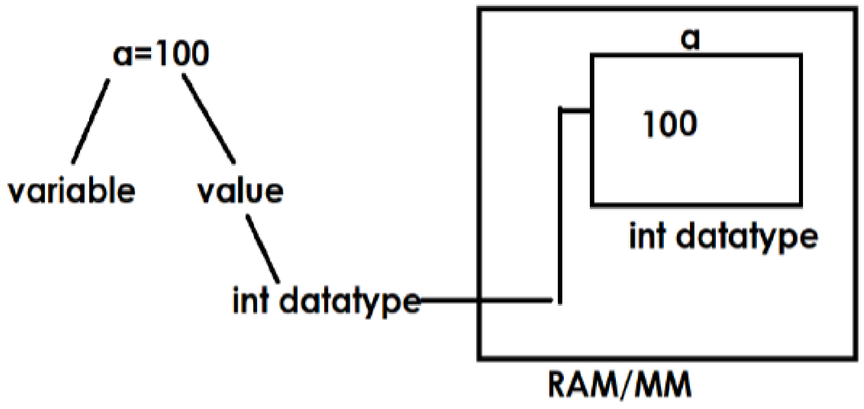
Integer value is numeric value without decimal point.

Example: whole numbers, even numbers, odd numbers,…

In python integer values are represented in memory using “int” data type.

Creating variables

What is variable?
Variable is an identifier which is used to identify value
The variable is named memory location.
Variables are created to represent data in memory.
Syntax: **variablename=value/literal**



Example: Integer Data Type (`int`)

```
>>> a = 10
>>> b = 20
>>> a
10
```

More Examples of `int`

```
>>> student_rollno = 1
>>> account_number = 1000000099999
>>> otpno = 3344
```

Type Conversion to Integer

```
>>> seatno = int(5)
>>> x = int(1.5)
>>> x
1
```

Note:

- `int(1.5)` converts the float value to integer by **removing the decimal part**, not rounding.
- Python supports **very large integers** without any limit.

A. Integer Literal

Integer Literals and `int` Data Type

What is an Integer Literal or Value?

An **integer literal** is a numeric value **without a decimal point**.
Integer values or literals can be:

- Whole numbers
- Even numbers
- Odd numbers

In Python, integer values are represented in memory using the **`int` data type**.

Dynamic Typing in Python

Python is a **dynamically typed programming language**.
The type of a variable is **not fixed** and changes based on the value assigned to it.

Syntax

```
variable_name = value / literal
```

Python and Object-Oriented Programming

Python is an **object-oriented programming language**.
In object-oriented programming:

- Data is represented as **objects**
- Every data type is a **class**

Example

```
>>> a = 50

>>> a

50

>>> type(a)

<class 'int'>


>>> rollno = 1
>>> otpno = 4455
>>> type(rollno)
<class 'int'>
>>> type(otpno)
<class 'int'>
```

Integer Number Systems in Python

In Python, integer values are represented in **four formats**:

1. Decimal Integer

2. Octal Integer

3. Hexadecimal Integer

4. Binary Integer

These are called **number systems**.

A **number system** defines a set of rules and regulations for representing numbers in computer science.

Based on the **base (radix)** of the integer value, Python supports **four integer formats**.

A base represents the total number of digits used to construct a number.

1.Decimal Integer

An **integer value with base 10** is called a **decimal integer**.

Characteristics of Decimal Integers

- Constructed using digits **0 to 9**
- Can have a **+ or − sign**
- **Should not start with 0**
- Default representation of integers in Python is **decimal**

Examples

```
>>> n1 = 125
>>> n1
125

>>> n2 = -450
>>> n2
-450
```

Invalid Decimal Integers

```
>>> n3 = 4a2
SyntaxError: invalid decimal literal

>>> n4 = 030
SyntaxError: leading zeros in decimal integer
literals are not permitted;
use an 0o prefix for octal integers
```

More Examples

```
>>> rollno = 123
>>> otp = 5566
>>> accno = 1111111222223333
```

Checking Data Type

```
>>> type(rollno)
<class 'int'>
>>> type(otp)
<class 'int'>
>>> type(accno)
<class 'int'>
```

2.Octal Integer

An integer value with **base 8** is called an **octal integer**.

- Octal numbers use digits in the range **0 to 7**
- An octal integer is prefixed with **`0o`** or **`0O`**

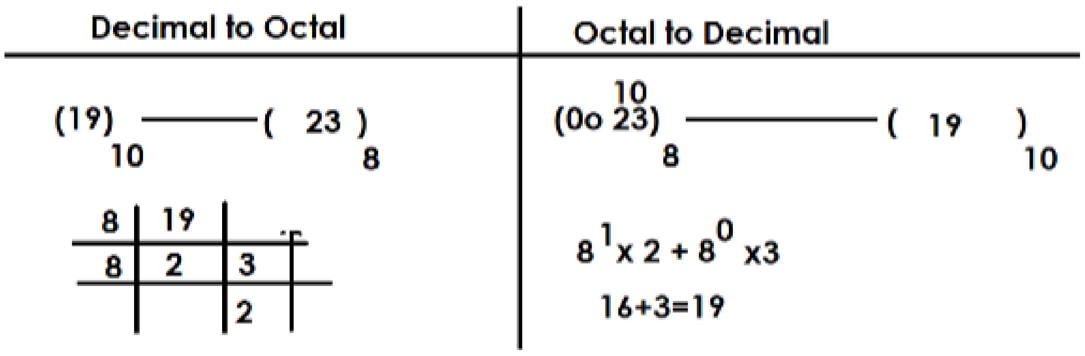
Usage of Octal Integers in Application Development

1. **Operating Systems** (File permissions)
2. **IP Addressing** (historical / low-level representations)

Python Examples

```
>>> a = 0o9
```

```
SyntaxError: invalid digit '9' in octal literal

>>> b = 0o128
SyntaxError: invalid digit '8' in octal literal

>>> c = 0o125
>>> c
85

>>> d = 0o12
>>> d
10

>>> type(d)
<class 'int'>

>>> e = 0o456
>>> type(e)
<class 'int'>
```



Decimal to Octal
Octal to Decimal
$(19)_{10} \longrightarrow (23)_8$
$(0o\ 23)_8 \longrightarrow (19)_{10}$
$8^1 \times 2 + 8^0 \times 3$
16+3=19

3.Hexadecimal Integer

An integer value with **base 16** is called a **hexadecimal integer**.

- This integer is prefixed with **`0x`** or **`0X`**
- Hexadecimal values use digits in the range **0–9** and **a–f / A–F**
- Larger values are commonly represented using hexadecimal format

Usage of Hexadecimal Integers in Application Development

1. **Color values**
2. **Memory addresses**
3. **Register addresses**
4. **Images, Audio, Video**

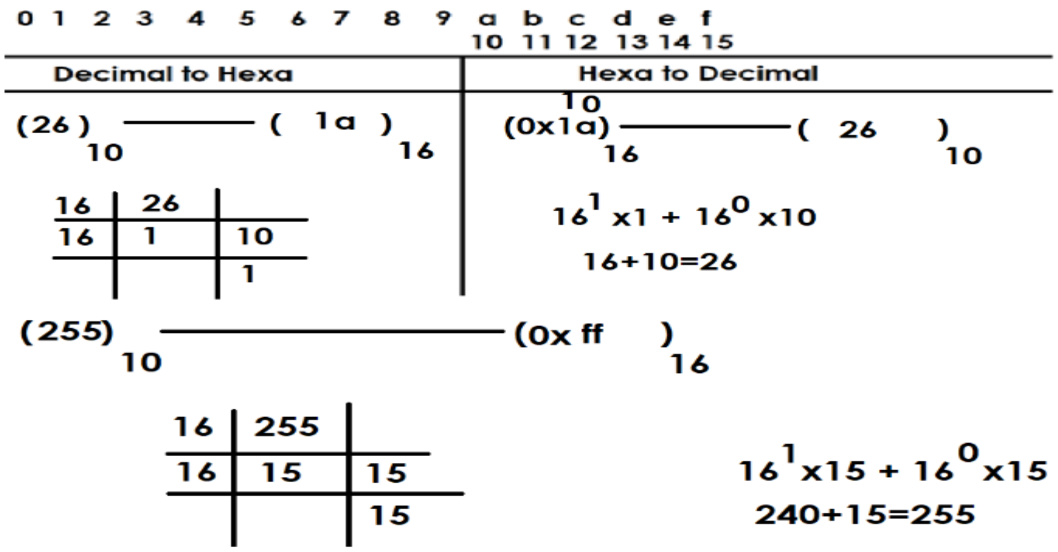
Hexadecimal Digits Mapping

```
0   1   2   3   4   5   6   7   8   9   a   b   c   d   e   f
0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
```

Python Examples

```
>>> n1 = 0xa
>>> type(n1)
<class 'int'>

>>> n2 = 56
>>> type(n2)
<class 'int'>

>>> n3 = 0o25
>>> type(n3)
<class 'int'>
```

```
>>> n1 = 0xa
>>> n1
10

>>> n2 = 0xd
>>> n2
13

>>> n3 = 0xff
>>> n3
255

>>> n4 = 0x12
>>> n4
18
>>> n5 = 0xg
SyntaxError: invalid hexadecimal literal
>>> n6 = 0xbad
>>> n6
2989
```



0 1 2 3 4 5 6 7 8 9 a b c d e f
10 11 12 13 14 15
Decimal to Hexa
Hexa to Decimal
$(26)_{10} \longrightarrow (1a)_{16}$
$(0x1a)_{16} \longrightarrow (26)_{10}$
10
16 | 26
16 | 1 | 10
1
$16^1 x1 + 16^0 x10$
16+10=26
$(255)_{10} \longrightarrow (0x\ ff)_{16}$
16 | 255
16 | 15 | 15
15
$16^1x15 + 16^0x15$
240+15=255

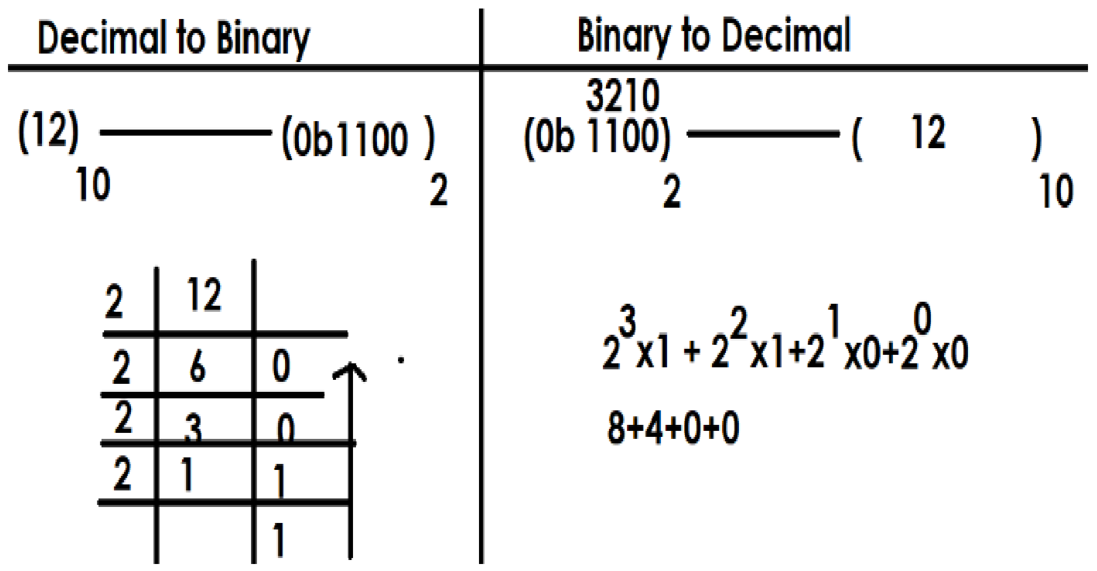
4.Binary Integer

An integer value with base 2 is called a **binary integer**.
This integer value is created using digits **0 and 1**.
This integer is prefixed with **0b** or **0B**.

In application development, binary integers are used in:

1. Memory representation
2. Embedded applications / Low-level programming

```
>>> a = 0b12
SyntaxError: invalid digit '2' in binary literal

>>> b = 0b101
>>> b
5

>>> c = 0B1010
>>> c
10

>>> d = 0b1100
>>> d
12

>>> e = 101
>>> e
101
```



Decimal to Binary
$(12)_{10}$ — $(0b1100)_2$
Binary to Decimal
3210
$(0b\,1100)_2$ — $(12)_{10}$
$2^3 \times 1 + 2^2 \times 1 + 2^1 \times 0 + 2^0 \times 0$
8+4+0+0

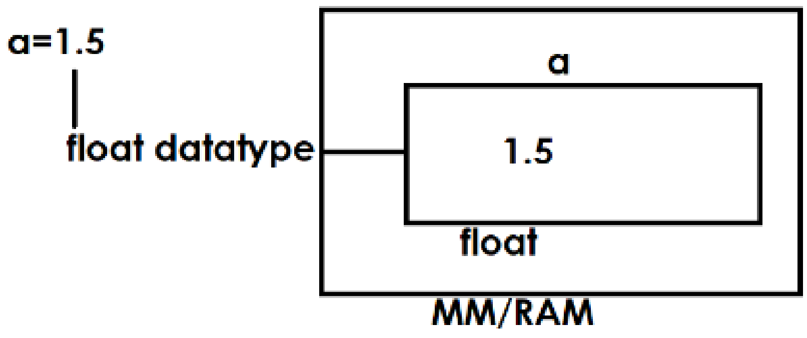
Size of int Data Type in Python

The size of the `int` data type in Python is **dynamic**.
Memory is reserved based on the **value size**.

```
>>> x = 99999
>>> import sys
>>> sys.getsizeof(x)
28

>>> y = 9999999999999999999999
>>> sys.getsizeof(y)
36

>>> z =
9999999999999999999999999999999999999999999999
>>> sys.getsizeof(z)
48
```

B. Float datatype and Float literal

float literal or value is numeric value with decimal part or decimal point.
In python float data type is used for representing float values in memory.



Float Data Type in Python

In Python, float values or literals are represented in **two formats**:

1. **Fixed Notation** or **Standard Notation** or **Decimal Notation**

2. **Exponent Notation** or **Scientific Notation**

```
>>> price = 45.65
>>> type(price)
<class 'float'>

>>> qty = 5
>>> type(qty)
<class 'int'>
```

1. Fixed Notation (Standard / Decimal Notation)

In this format or notation, a sequence of digits is followed by a **decimal point**.

Examples:
1.25, 456.35, 78923.0

This is the **default form** of representing float values or literals.

```
>>> mean = 25.78
>>> type(mean)
<class 'float'>

>>> median = 2.5
>>> type(median)
<class 'float'>

>>> strike_rate = 200.20
>>> type(strike_rate)
<class 'float'>

>>> avg = 78
>>> type(avg)
<class 'int'>
```

---

2. Exponential Form (Scientific Notation)

A **significant part** (integer or fractional) followed by an **exponent part** introduced by the letter e or E.
The exponent indicates a **power of 10**.
If the value is very large, it is represented in scientific notation.

```
>>> a = 5e1
>>> a
```

```
50.0
>>> type(a)
<class 'float'>

>>> b = 123e-1
>>> b
12.3
>>> type(b)
<class 'float'>

>>> c = 1234.45e2
>>> c
123445.0

>>> d = 1234.45e-2
>>> d
12.3445

>>> a = 10.0
>>> b = 1e1

>>> a
10.0
>>> b
10.0

>>> type(a)
<class 'float'>
>>> type(b)
<class 'float'>
```

How to Find Information About Float

```
>>> import sys
>>> sys.float_info
sys.float_info(
    max=1.7976931348623157e+308,
    max_exp=1024,
    max_10_exp=308,
    min=2.2250738585072014e-308,
    min_exp=-1021,
    min_10_exp=-307,
    dig=15,
    mant_dig=53,
    epsilon=2.220446049250313e-16,
    radix=2,
    rounds=1
)

>>> a = 1.7976931348623157e+308
>>> a
1.7976931348623157e+308

>>> b = 1.7976931348623157e+309
>>> b
inf

>>> type(b)
<class 'float'>
```

C. Complex Literal and Complex Data Type

A complex number is a numeric value with **two values**:

1. Real
2. Imaginary

Real and imaginary values are of type **float**.

In Python, complex numbers are represented in memory using the **complex data type**.

Syntax:

```
a + bj
```

- a → real value
- b → imaginary value

The imaginary value is suffixed with **`j` or J**.

Example:

```
>>> import math
>>> math.sqrt(9)
3.0

>>> math.sqrt(-9)
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    math.sqrt(-9)
ValueError: expected a nonnegative input, got -9.0
```

Using cmath module:

```
>>> import cmath
>>> cmath.sqrt(-9)
3j
```

Complex number examples:

```
>>> c1 = 1 + 2j
>>> c1
(1+2j)

>>> c1.real
1.0

>>> c1.imag
2.0

>>> c2 = 3j
>>> c2
3j

>>> c2.real
0.0


>>> c2.imag
3.0

>>> type(c2)
<class 'complex'>

>>> c3 = 3
>>> type(c3)
<class 'int'>
```

```
>>> c4 = 3 + 0j
>>> type(c4)
<class 'complex'>

>>> c4.real
3.0

>>> c4.imag
0.0
```

D. Bool Datatype and Boolean Values / Literals
In Python, boolean values are represented using two keywords:

1. True

2. False

These boolean values are represented in memory using the bool data type.

Examples:

```
>>> rollno = 123
>>> type(rollno)
<class 'int'>

>>> fee = 9000.0
>>> type(fee)
<class 'float'>

>>> fee_paid = True
>>> type(fee_paid)
<class 'bool'>

>>> a = True
>>> a
```

```
True

>>> type(a)
<class 'bool'>

>>> b = False
>>> b
False

>>> type(b)
<class 'bool'>

>>> 10 > 5
True

>>> 10 > 20
False

>>> 10 + 20
30

>>> seatno = 1
>>> seat_reserved = False

>>> type(seatno)
<class 'int'>

>>> type(seat_reserved)
<class 'bool'>
```

E. NoneType and None Value / Literal
None is a keyword which represents no value or missing value.

None value in memory is represented using the NoneType datatype.

Examples:

python
Copy code
```
>>> rollno = 123
>>> name = None
>>> course = None

>>> type(rollno)
<class 'int'>

>>> type(name)
<class 'NoneType'>

>>> type(course)
<class 'NoneType'>

>>> name
>>> course
```
Input and Output Statements in Python
Programming elements are three:

Input

Process

Output