Introduction to Python

IDE: An IDE for Python is a software tool that provides a code editor, debugger, and run environment in one place. It helps write, test, and debug Python programs easily and efficiently.

Example - Anaconda, google collab

Variable: A variable is a name used to store data in a program. It acts like a container that holds a value which can change during execution.
Example: Raj = ‘Ratnajit’. Here Raj is a variable, and ‘Ratnajit’ is the value which is stored inside the Raj variable. Ratnajit is a string literal.

Rules for Variables (Python)

1. A variable name can contain **letters, digits, and underscore** (`_`).
2. A variable name **must start** with a letter or underscore — **not** with a number.
3. A variable name **cannot contain spaces**.
4. A variable name **cannot be a Python keyword** (like `for`, `if`, `while`, etc.).
5. Variable names are **case-sensitive** (`age ≠ Age`).
6. Use descriptive names or **CamelCase / snake_case** for clarity.

Literals: Literals are fixed constant values written directly in a program. They represent data like numbers, strings, or booleans that do not change.
Example: 5, `"Hello"`, `True`.

Keyword: Keywords are special reserved words in Python that have a predefined meaning and purpose. They cannot be used as variable names or identifiers.
Example: `['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']`

import keyword
print(len(keyword.kwlist))
35

Note:
Python is a dynamic language because variable types are determined at runtime, whereas C and C++ are static languages because variable types must be declared before use.

Int Raj = 12 ❌

Raj = 12 ✓

Python Vs Other programming languages

- Interpreted Language (Python):
  Runs line by line → Interpreter executes each line → Slower, flexible.
- Compiled Language (C, C++, Java):
  Code is compiled all at once → Creates an executable → Faster.

Data Types in Python

1. Numeric Types

- **int** → whole numbers (e.g., `10`, `-5`)
- **float** → decimal numbers (e.g., `3.14`, `2.0`)
- **complex** → complex numbers (e.g., `3+5j`)

---

2. String Type

- **str** → text data (e.g., `‘Ratnajit’`, `‘Python’`)

---

3. Boolean Type

- **bool** → `True` or `False`

---

4. Sequence Types

- **list** → ordered, changeable, allows duplicates (e.g., `[1, 2, 3]`)
- **tuple** → ordered, unchangeable (immutable), duplicates allowed (e.g., `(1, 2, 3)`)
- **range** → sequence of numbers (e.g., `range(5)`)

---

5. Mapping Type

- **dict** → key-value pairs (e.g., `{‘name’: ‘Raj’, ‘age’: 28}`)

---

6. Set Types

- **set** → unordered, unique elements (e.g., `{1, 2, 3}`)
- **frozenset** → immutable set

---

7. None Type

- **NoneType** → represents no value (e.g., `None`)

List Examples

Example 1: Raj = ['Raj', 12, '&8123', 'Programming', 18.9]
print(Raj)
print(type(Raj))

```
['Raj', 12, '&8123', 'Programming', 18.9]
<class 'list'>
```

Example 2: _Raj = list('python')
print(_Raj)
print(type(_Raj))
['p', 'y', 't', 'h', 'o', 'n']
<class 'list'>

`list('Raj')`

`list()` can take **one argument**, and that argument must be an **iterable**.

A **string is an iterable**, it can be split into characters.

`Output: ['R', 'a', 'j']`
Because Python loops through the string and adds each character to the list.
Note: **We must pass a string when we are using the function list.**

```
# We can pass list inside a list
mixed = [12, 16, [23,10, 'Raj'], 56, 'Python']
for i in mixed[2]:
   print(f'{type(i)}: {i}')
```

Output:
```
<class 'int'>: 23
<class 'int'>: 10
<class 'str'>: Raj
```

List:

`[12, 16, [23, 10, 'Raj'], 56, 'Python']`

Index Positions:

Index	Value
0	12
1	16
2	[23, 10, 'Raj']
3	56
4	'Python'

Indexes inside the nested list (index 2)

Nested list: `[23, 10, 'Raj']`

Index	Value
2[0]	23
2[1]	10
2[2]	‘Raj’

Reverse Indexing:

List : `[12, 16, [23, 10, 'Raj'], 56, 'Python']`

Index	Value
-1	‘Python’
-2	56
-3	[23,10,’Raj’]

-4 16

-5 12

# We can use the + operator to combine two list slices

```
new_list = [12,14,67,89,90,78,789,34,86,17]
new_list[0:2] + new_list[4:6]
```

Output : [12, 14, 90, 78]

Properties of List in Python

1. **Ordered**
    - List elements maintain the same order in which they were inserted.
    - Indexing is possible (positive & negative).

2. **Mutable**
    - Lists can be changed after creation.
    - It is mutable:
        1. We can **replace**
        2. We can **insert**
        3. We can **Remove**

3. Heterogeneous

- A list can store different data types:
- Example: `[10, 20.5, "Python", [1,2,3]]`

4.Allows Duplicate Values:

- Lists can contain repeated elements.

  Example: `[10, 20, 10, 30]`

5.Supports Slicing

- We can extract part of a list using slicing:

```
List[start:stop:step]
```

6. Nesting Allowed

- A list can contain other lists (2D list / nested list).

NOTE on Square Brackets [ ] in Python

1. [ ] used ALONE → Creates a LIST

Example:

```
my_list = []
```

This means we are creating a **list**.

---

2. [ ] used AFTER a variable → Selection / Indexing / Slicing

Example:

```
my_list[0]

my_list[1:3]
```

Here, the square brackets are **not a list**.
They are used for **data selection**, **indexing**, or **slicing** from:

- lists
- tuples
- strings
- dictionaries
- sets (only with loops, not direct indexing)

Concept	Meaning	Syntax	Example	Output
Indexing	Selecting **one item** by its position	`variable[index]`	`nums[2]`	value at index 2
Slicing	Selecting **multiple items** (a range)	`Variable[start:stop + 1:step]`	`nums[1:4]`	values from index 1 to 3

Example 1: Indexing (one value)

list1 = [12,13,’Python’, 9012]
list1[0]
Output = 12

Example 2: Slicing (range of values)
list1 = [12,13,’Python’, 9012]
list1[0:3]
Output = [12,13, ‘Python’]

Example 3: Data Selection

student = {'name':'Raj', 'age':28, 'course':'Python', 'marks':92, 'city':'Kolkata'}
student[‘name’]
Output = ‘Raj’

Sets

Properties of Sets in Python

1. **Unordered**
    - Set elements do not follow any indexing or order.
    - You cannot access items using indexes like set[0].

2. **No Duplicate Values**
    - A set automatically removes duplicates.
    - Example: {10, 20, 10} → {10, 20}

3. **Mutable**
    - **We can add or remove elements using:**

        ★ **add()**

        ★ **update()**

        ★ **remove()**

        ★ **discard()**

4. Heterogeneous
- A list can store different data types:
- Example: **[10, 20.5, "Python", [1,2,3]]**

5. Unindexed

- Since sets are unordered, elements have **no index positions**.

6. Supports Mathematical Set Operations

- Union → set1 | set2
- Intersection → set1 & set2
- Difference → set1 - set2
- Symmetric difference → set1 ^ set2

1. Union → `set1 | set2`

Use: Combines all elements from both sets, removing duplicates.

---

2. Intersection → `set1 & set2`

Use: Returns only the common elements present in both sets.

---

3. Difference → `set1 - set2`

Use: Returns elements that are in set1 but not in set2.

---

4. Symmetric Difference → `set1 ^ set2`

Use: Returns elements that are unique to each set (not common).

Difference between .discard() and .remove()

discard() does not gives an error if the value is not found whereas remove() gives an error.

Properties of Dictionary in Python

1. Unordered (before Python 3.7) / Ordered (Python 3.7+)

- **From Python 3.7 onwards, dictionaries maintain insertion order.**
- **Traditionally they were considered unordered.**

---

2. Mutable

- **You can add, update, or remove key-value pairs after creation.**

---

3. Stores Key–Value Pairs

Data is stored in the form of: key : value

Example: {‘Raj’: 28}

Key = Raj

Value = 28

---

4. Keys Must Be Unique

- **Duplicate keys are not allowed.**
- **If you use the same key twice, the last value overwrites the previous one.**

---

5. Keys Must Be Immutable

- **Keys can be: int, float, string, tuple**
- **Keys cannot be: list, set, dictionary**

---

6. Values Can Be of Any Data Type

- **Values may be duplicate.**

- **Values may be list, set, dict, tuple, anything.**

---

7. Dynamic

- **Size can increase or decrease at runtime.**

---

8. Supports Nested Dictionaries

- **A dictionary can contain another dictionary.**

Note on Dictionary and Set (in simple words)

Dictionaries are somewhat similar to sets because both are unordered, mutable, and store unique items.
However, in a set we cannot select a particular value because sets have no indexing.

To overcome this, dictionary uses key–value pairs. Each value is connected to a key, and we can access any specific value by using its key name.

So, a dictionary is like an improved version of a set where:

- items are stored as key : value
- keys are unique
- we can easily get any value using its key

Tuples

Properties of Tuples in Python

1. Ordered

- Tuples maintain the order of elements.
- Indexing is allowed.

---

2. Immutable

- We cannot modify, add, or remove elements after creation.

---

3. Heterogeneous

- A tuple can store values of different data types
  (int, float, string, list, tuple, etc.).

---

4. Allows Duplicate Values

- Same value can appear more than once.

---

5. Faster than Lists

- Because they are immutable, access is slightly faster.

---

6. Can Be Used as Dictionary Keys

- A tuple can be used as a dictionary key (if all elements inside it are immutable).
- Example: my_dict = {('Raj', 'Ratnajit'): 10}

  my_dict[('Raj', 'Ratnajit')]

  Output = 10

---

7. Supports Indexing and Slicing

- Same as lists, we can do:

  t[0], t[-1], t[1:3]

Type Casting (Type Conversion)

Type casting is the process of converting one data type into another data type in Python.

–---

String

Properties of String in Python

1. Immutable

- **You cannot modify a string after creation.**
- **No adding, removing, or changing characters directly.**

---

2. Ordered / Indexed

- **Each character has a fixed position.**
- **Positive and negative indexing is allowed.**

Example:
`"Python"`
Indexes → `0 1 2 3 4 5`
Reverse → `-6 -5 -4 -3 -2 -1`

---

3. Iterable

- You can loop through each character using a `for` loop.

---

4. Heterogeneous?

- No. Strings store only characters (text).
- They cannot store mixed data types.

---

5. Allows Duplicate Characters

- Characters can repeat.
  Example: `"Programming"`

---

6. Supports Slicing

You can extract substrings using slicing:

```
word[0:6]
```

-

---

7. Supports Many Built-in Methods

- Examples:
    - `.upper()`
    - `.lower()`
    - `.find()`
    - `.replace()`
    - `.split()`

---

8. Stored in Unicode

- **Supports multiple languages and special characters.**