Python Basics

Tuesday, June 10, 2025 1:14 PM

Print Output:

- print("hello")
- print(57)
- print(True)
- print("hello",45,True,0)
- print("hello",45,True,0,sep="---")
- Print("hello","hii", end="\n")

Datatypes:

Different Kinds of Python Data Types

There are different types of data types in Python. Some built-in Python data types are:

1. Numeric data types: int, float, complex
2. String data types: str
3. Sequence types: list, tuple, range
4. Binary types: bytes, bytearray, memoryview
5. Mapping data type: dict
6. Boolean type: bool
7. Set data types: set, frozenset

Basic Built-in Data Types

Data Type	Description	Example
int	Whole numbers	`x = 10`
float	Decimal numbers	`pi = 3.14`
str	Text / String	`name = "Alice"`
bool	True or False	`is_active = True`
list	Ordered collection (changeable)	`fruits = ["apple", "banana"]`
tuple	Ordered collection (unchangeable)	`points = (1, 2)`
dict	Key-value pairs	`person = {"name": "Tom", "age": 25}`
set	Unordered, unique values	`nums = {1, 2, 3}`
NoneType	Represents no value	`x = None`

Binary Types: bytes, bytearray, memoryview

These are used to handle binary data, like files, images, or when working with network or hardware communication.

Bytes: Immutable (can't change it after creation),Stores binary data

```
b = bytes([65, 66, 67])

print(b)  # Output: b'ABC'

print(b[0])     # Output: 65
```

Bytearray: Mutable version of bytes, You can modify its content

```
ba = bytearray([65, 66, 67])

ba[0] = 68

print(ba)       # Output: bytearray(b'DBC')
```

Memoryview:

```
data = bytearray(b"Hello")

mv = memoryview(data)

print(mv[0])        # Output: 72 (ASCII of 'H')
```

```
print(mv[1:4].tobytes())  # Output: b'ell'
```

Set Types: set, frozenset

set: Unordered collection of unique elements, Mutable (can add/remove items)

```
s = {1, 2, 3, 3}
print(s)      # Output: {1, 2, 3} — duplicates removed

s.add(4)
```

Frozenset: Immutable version of a set (can't add/remove items)

```
fs = frozenset([1, 2, 3])

# fs.add(4)  Error: cannot modify frozenset
print(fs)
```

In Python, all data types are either:

- **Mutable** → **Can be changed** after creation
- **Immutable** → **Cannot be changed** after creation

Mutable Data Types

- You **can modify** the data (add, remove, change items).

Type	Example
list	[1, 2, 3]
dict	{"a": 1}
set	{1, 2, 3}
bytearray	bytearray([65, 66])

Immutable Data Types

You **cannot change** the data once created. If you try to change, it creates a **new object**.

Type	Example
int	5
float	3.14
str	"hello"
tuple	(1, 2)
frozenset	frozenset([1, 2])
bool	True, False
bytes	b"data"

Python Keywords:

Python keywords are the words that are reserved. That means you can't use them as name of any entities like variables, classes and functions.

```
help> keywords

Here is a list of the Python keywords.  Enter any keyword to get more help.

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

Python Identifiers:

Python Identifier is the name we give to identify a variable, function, class, module or other object. That means whenever we want to give an entity a name, that's called identifier.

Rules for writing identifiers:

- Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
- Starting an identifier with a single leading underscore indicates the identifier is private.

- If the identifier starts and ends with two underscores, than means the identifier is language-defined special name.
- While c = 10 is valid, writing count = 10 would make more sense and it would be easier to figure out what it does even when you look at your code after a long time.
- Multiple words can be separated using an underscore, for example this_is_a_variable.

Use case of datatypes in real word:

1. List

Definition: An ordered, mutable collection that allows duplicate values. Use lists when you want to collect, modify, and process sequential data.

Case 1: Storing ordered records (basic example)

```
students = ["Alice", "Bob", "Charlie"]
```

Case 2: Appending new data to a dataset (e.g., log entries)

```
log_entries = []
log_entries.append("2025-06-11: Login Success")
log_entries.append("2025-06-11: File Uploaded")
```

Case 3: Storing a column of data from a CSV file

```
import csv
with open("sales.csv") as file:
reader = csv.reader(file)
prices = []
for row in reader:
 prices.append(float(row[2]))       # assuming price is in column 3
```

2. Tuple

Definition: An ordered, immutable collection. Useful for fixed-size, fixed-content data. Tuples are great for fixed, read-only data, or to be used as keys in dicts.

Case 1: Representing coordinates or fixed attributes

```
location = (28.6139, 77.2090)
```

Case 2: Returning multiple values from a function (structured data)

```
def get_student():
      return ("A101", "Alice", 14)
roll, name, age = get_student()
```

Case 3: Using tuples as keys in a dictionary (composite keys)

```
sales = { ("India", "2025-06-11"): 1500, ("UAE", "2025-06-11"): 2100 }
```

3. Dictionary

Definition: A mutable collection of key-value pairs with unique keys. Use dictionaries for keyed data access, data modeling, and lookup efficiency.

Case 1: Storing structured records with fields

```
employee = {"id": "E123", "name": "Raj", "dept": "HR"}
```

Case 2: Grouping multiple records for fast lookup by ID

```
employees = { "E101": {"name": "Alice", "age": 25}, "E102": {"name": "Bob", "age": 28} }
```

Case 3: Counting occurrences (e.g., word frequency)

```
text = "apple banana apple"
word_count = {}
for word in text.split():
    word_count[word] = word_count.get(word, 0) + 1
```

4. Set

Definition: An unordered, mutable collection of unique items. Sets are ideal when uniqueness matters and for fast membership checks.

Case 1: Removing duplicate values from a list

```
emails = ["a@example.com", "b@example.com", "a@example.com"]
unique_emails = set(emails)
```

Case 2: Performing set operations (intersection, union, etc.)

```
trained_skills = {"Python", "SQL", "Excel"}
required_skills = {"SQL", "Power BI", "Excel"}
matching = trained_skills & required_skills          # {'SQL', 'Excel'}
```

Case 3: Checking for data membership efficiently

```
blacklist = {"abc@gmail.com", "xyz@gmail.com"}
if "abc@gmail.com" in blacklist:
    print("Access denied")
```

Type	Mutable	Ordered	Duplicates Allowed	Ideal For
List	Yes	Yes	Yes	Dynamic data collections
Tuple	No	Yes	Yes	Immutable records, function returns
Dict	Yes	Keys only	No (keys)	Structured data, fast lookup
Set	Yes	No	No	Unique items, fast lookups