Detailed Guide to Python Data Types

Python is a high-level, interpreted programming language known for its readability and simplicity. It's widely used for web development, data analysis, artificial intelligence, and automation.

Built-in Data Types in Python
Python comes with several built-in data types, which can be broadly categorized as follows:

a) Numeric Types: int, float, complex
b) Sequence Types: str, list, tuple, range
c) Mapping Type: dict
d) Set Types: set, frozenset
e) Boolean Type: bool
f) None Type: NoneType

Let's delve into each of these in detail.

Numeric Types
Numeric types store numerical values. They are immutable, meaning their value cannot be changed after creation.

int (Integer): - Represents whole numbers (positive, negative, or zero) without a fractional part. Integers in Python have arbitrary precision, meaning they can be as large as your system's memory allows.

```
a =14
print(a,':',type(a))
```
```
[2] ✓ 0.0s
14 : <class 'int'>
```

float (Floating-Point Number):-Represents real numbers with a decimal point or in exponential form. They are implemented using double-precision floating-point numbers.

```
scientific_notation = 1.23e-5
print(scientific_notation,':',type(scientific_notation))
```
```
[3] ✓ 0.0s
1.23e-05 : <class 'float'>
```

Complex (Complex Number):-Represents complex numbers, which have a real and an imaginary part. The imaginary part is denoted by j or J.

```
z1 = 3 + 4j
print(z1,':',type(z1))
```
[4] ✓ 0.0s

```
(3+4j) : <class 'complex'>
```

Sequence Types

Sequence types are ordered collections of items. They allow access to items by their index.

str (String) :-

Represents a sequence of Unicode characters. Strings are used for text data. They can be defined using single quotes ('...'), double quotes ("..."), or triple quotes ('''...''' or """...""") for multi-line strings.

Key Characteristics:

- Ordered collection of characters.
- Supports indexing (accessing individual characters) and slicing (substrings).

```
message = 'Hello, IT FUTURE TECH !'
print(message,':',type(message))
```
[6] ✓ 0.0s

```
Hello, IT FUTURE TECH ! : <class 'str'>
```

Common Methods: len(), lower(), upper(), strip(), split(), join(), find(), replace(), startswith(), endswith().

list (List) :-

An ordered, mutable collection of items. Lists can contain items of different data types.

Key Characteristics:

- Ordered (maintains insertion order).
- Allows duplicate members.
- Supports indexing and slicing.

```
my_list = [1, "IT", "Future", True]
print(my_list,':',type(my_list))
```
[8] ✓ 0.0s

```
[1, 'IT', 'Future', True] : <class 'list'>
```

Common Methods: len(), append(), extend(), insert(), remove(), pop(), clear(), index(), count(), sort(), reverse().

tuple (Tuple) :-

An ordered, immutable collection of items. Tuples are similar to lists but cannot be changed after creation.
Key Characteristics:

- Ordered (maintains insertion order).
- Allows duplicate members.
- Supports indexing and slicing.

```
my_tuple = ("IT", "Future", "Tech")
print(my_tuple,':',type(my_tuple))
```
```
[10] ✓ 0.0s
```
```
('IT', 'Future', 'Tech') : <class 'tuple'>
```

range (Range) :-

Represents an immutable sequence of numbers, commonly used for looping a specific number of times in for loops. It is a memory-efficient way to represent sequences.
Key Characteristics:

- Generates numbers on the fly (lazy evaluation), not all at once. o Syntax: range(stop), range(start, stop), range(start, stop, step).

```
numbers = range(5)
for i in numbers:
    print(i, end=" ")
print("\n",type(numbers))
```
```
[14] ✓ 0.0s
```
```
0 1 2 3 4
 <class 'range'>
```

Mapping Type

dict (Dictionary) :-

An unordered (in Python versions before 3.7, ordered from 3.7+), mutable collection of key-value pairs. Each key must be unique and immutable (e.g., strings, numbers, tuples). Values can be of any data type.
Key Characteristics:

- Stores data in key: value pairs.
- Keys must be unique and hashable (immutable).
- Values can be duplicates and mutable.

- Efficient lookup by key.

```
person = {"name": "ABC", "age": 27, "city": "Mumbai"}
print(person["name"])
person["age"] = 26
person["email"] = "ABC@itfuturetech.com"
print(type(person))
print(person)
```

[18] ✓ 0.0s

```
ABC
<class 'dict'>
{'name': 'ABC', 'age': 26, 'city': 'Mumbai', 'email': 'ABC@itfuturetech.com'}
```

Common Methods: len(), keys(), values(), items(), get(), pop(), popitem(), update(), clear().

Set Types

Set types are unordered collections of unique items. They are primarily used for membership testing and eliminating duplicate entries.

set (Set):-

An unordered, mutable collection of unique and hashable items.

Key Characteristics:

- Does not allow duplicate members.
- Items must be hashable (immutable types like numbers, strings, tuples). Lists and dictionaries cannot be set members.
- Supports mathematical set operations like union, intersection, difference, and symmetric difference.

```
my_set = {1, 2, 3, 2, 4} # Duplicates are automatically removed
print(my_set)
```

[19] ✓ 0.0s

```
{1, 2, 3, 4}
```

Common Methods: len(), add(), remove(), discard(), pop(), clear(), union(), intersection(), difference(), issubset(), issuperset().

frozenset (Frozen Set):-

An unordered, immutable collection of unique and hashable items.

Key Characteristics:

- Similar to set, but once created, its elements cannot be changed.
- Because it's immutable, a frozenset can be used as a key in a dictionary or an element in another set.

```
my_frozenset = frozenset([1, 2, 3, 2])
print(my_frozenset)
```
[20] ✓ 0.0s

```
frozenset({1, 2, 3})
```

Boolean Type

bool (Boolean) :- Represents truth values: True or False.
Key Characteristics:

- Used in conditional statements (if, while) and logical operations (and, or, not).
- True evaluates to 1 and False evaluates to 0 in numeric contexts.

```
is_active = True
print(type(is_active))
```
[21] ✓ 0.0s

```
<class 'bool'>
```

None Type

NoneType (None) :- Represents the absence of a value or a null value. It is a unique constant object of type NoneType.
Key Characteristics:

- Often used to initialize a variable that will later hold a value, or as a default return value for functions that don't explicitly return anything.
- None is not the same as 0, False, or an empty string/list.

```
result = None
print(type(result))
```
[22] ✓ 0.0s

```
<class 'NoneType'>
```

Type Conversion (Type Casting)

Python provides built-in functions to convert values from one data type to another. This is also known as type casting.

- int(): Converts to an integer.
- float(): Converts to a floating-point number.
- str(): Converts to a string.
- list(): Converts to a list.
- tuple(): Converts to a tuple.
- set(): Converts to a set.
- dict(): Converts to a dictionary (from a sequence of key-value pairs).
- bool(): Converts to a boolean. Examples: # To int num_str = "123" num_int = int(num_str)

Conclusion

Python's rich set of built-in data types provides powerful ways to represent and manipulate various kinds of information. Understanding their characteristics, especially mutability, and knowing how to convert between them, is crucial for writing robust, efficient, and maintainable Python code. Always choose the most appropriate data type for your specific needs to ensure your programs behave as expected.