Python Data Types

Understand the Values Your Programs Use

---

1. Why Data Types Matter

Imagine you are building a user profile for haas.dev.

You may need to store:

- A user's name
- Their age
- Their learning progress
- Their account status
- Their score
- Whether they completed a course

These values are not all the same kind of data.

```
name = "Hafsa"
age = 25
progress = 87.5
is_logged_in = True
```

Python needs to know what kind of value each variable contains because different types support different operations.

For example:

```
age = 25
print(age + 5)
```

This works because `age` contains an integer.

But:

```
name = "Hafsa"
print(name + 5)
```

This causes an error because Python cannot directly add an integer to a string.

Understanding data types helps you:

- store information correctly
- perform valid operations
- avoid common errors

- understand how Python behaves
- write predictable programs
- choose the right data structure later

Data types are one of the foundations of Python programming.

---

2. What Is a Data Type?

A **data type** describes what kind of value something is and what you can do with that value.

For example:

```
"Hafsa"
```

is a string.

```
25
```

is an integer.

```
87.5
```

is a floating-point number.

```
True
```

is a boolean.

Python is **dynamically typed**, which means you do not have to explicitly declare the type of a variable before assigning a value.

You can simply write:

```
age = 25
```

Python understands that `age` refers to an integer value.

---

3. Python's Common Built In Data Types

Some of the most important Python data types are:

Type	Example	Used For
`str`	`"Hafsa"`	Text
`int`	`25`	Whole numbers

`float`	`87.5`	Decimal numbers
`bool`	`True`	True/false values
`NoneType`	`None`	Absence of a value

Python also provides collection types such as:

- `list`
- `tuple`
- `set`
- `dict`

These will be covered in detail later.

For now, focus on understanding the basic value types.

---

4. Strings

A string represents text.

You create strings using quotes.

```
name = "Hafsa"
```

You can use single quotes:

```
name = 'Hafsa'
```

Or double quotes:

```
name = "Hafsa"
```

Both create a value of type `str`.

```
print(type(name))
```

Output:

```
<class 'str'>
```

4.1 What Can Strings Store?

Strings can contain:

- names
- sentences
- addresses
- usernames
- emails
- product names
- course titles
- symbols
- numbers represented as text

For example:

```
name = "Hafsa"
course = "Python"
email = "hafsa@example.com"
```

Even though an email contains symbols and a username may contain numbers, the complete value is still text.

4.2 Numbers Inside Strings

Consider:

```
age = "25"
```

This is **not** an integer.

It is a string containing the characters `2` and `5`.

Compare:

```
age1 = 25
age2 = "25"

print(type(age1))
print(type(age2))
```

Output:

```
<class 'int'>
<class 'str'>
```

This difference becomes important when performing calculations.

```
age1 = 25

print(age1 + 5)
```

Output:

```
30
```

But:

```
age2 = "25"

print(age2 + 5)
```

produces a `TypeError`.

Python does not automatically treat `"25"` as the number `25`.

---

5. Integers

The `int` type represents whole numbers.

Examples:

```
age = 25
score = 100
students = 50
temperature = -5
```

Integers can be:

- positive
- negative
- zero

Examples:

```
10
-10
0
```

5.1 Integer Calculations

Python can perform mathematical operations with integers.

```
a = 20
b = 5
```

```
print(a + b)
print(a - b)
print(a * b)
print(a / b)
```

Output:

```
25
15
100
4.0
```

Notice something important:

```
20 / 5
```

produces:

```
4.0
```

The result of `/` is a floating-point number.

5.2 Integer Division

Python also provides `//`.

```
print(20 // 6)
```

Output:

```
3
```

This performs floor division.

The remainder can be obtained using `%`.

```
print(20 % 6)
```

Output:

```
2
```

These operators become useful when solving programming problems.

---

6. Floating Point Numbers

The `float` type represents numbers containing a decimal component.

Examples:

```
price = 99.99
rating = 4.8
temperature = 36.5
progress = 87.5
```

Check the type:

```
progress = 87.5

print(type(progress))
```

Output:

```
<class 'float'>
```

6.1 Integer vs Float

Compare:

```
x = 10
y = 10.0
```

Their values look similar, but their types are different.

```
print(type(x))
print(type(y))
```

Output:

```
<class 'int'>
<class 'float'>
```

`10` is an integer.

`10.0` is a float.

---

7. Boolean Values

The `bool` type represents one of two logical values:

```
True
False
```

Boolean values are extremely important for decision making.

For example:

```
is_logged_in = True
```

or:

```
has_completed_course = False
```

You can check the type:

```
print(type(is_logged_in))
```

Output:

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

---

8. Booleans in Real Programs

Suppose haas.dev needs to determine whether a learner has completed a course.

```
completed = True

if completed:
    print("Certificate available")
```

Output:

```
Certificate available
```

Boolean values become especially important with:

- `if`
- `while`
- comparisons
- authentication
- permissions
- validation
- feature flags

For example:

```
age = 25

is_adult = age >= 18

print(is_adult)
```

Output:

```
True
```

The comparison produces a boolean value.

---

9. None

Python has a special value called `None`.

It represents the absence of a value.

```
result = None
```

Its type is:

```
print(type(result))
```

Output:

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

`None` does not mean:

- zero
- an empty string
- `False`

It specifically represents "no value" or "no value currently available."

9.1 Real World Example

Imagine a user has not uploaded a profile picture yet.

```
profile_picture = None
```

The variable exists, but there is currently no picture associated with it.

Later:

```
profile_picture = "profile.png"
```

Now it contains a string.

---

10. None vs 0 vs False vs Empty String

These values are different:

```
a = None
b = 0
c = False
d = ""
```

They represent different concepts.

Value	Meaning
`None`	No value
`0`	Numeric zero
`False`	Boolean false
`""`	Empty string

Do not treat them as interchangeable.

---

11. Checking a Value's Type

Python provides the built-in `type()` function.

```
name = "Hafsa"
age = 25
rating = 4.8
active = True

print(type(name))
print(type(age))
print(type(rating))
print(type(active))
```

Output:

```
<class 'str'>
<class 'int'>
<class 'float'>
<class 'bool'>
```

This is especially useful when learning and debugging.

---

12. Python Figures Out Types Automatically

In many programming languages, you may need to explicitly declare a variable's type.

Python generally does not require this.

You can write:

```
name = "Hafsa"
age = 25
```

Python determines the types from the assigned values.

```
name → str
age → int
```

You can even reassign a variable to a different type:

```
value = 25

print(type(value))

value = "Hafsa"

print(type(value))
```

Output:

```
<class 'int'>
<class 'str'>
```

This is one reason Python is called **dynamically typed**.

---

13. Dynamic Typing

Dynamic typing means that the type is associated with the value during program execution rather than requiring you to declare the variable's type beforehand.

For example:

```
data = 100
```

Here:

```
data → integer value
```

Later:

```
data = "Python"
```

Now:

```
data → string value
```

Python allows this.

However, changing a variable between unrelated types can sometimes make programs harder to understand.

For example:

```
data = 100
data = "Python"
data = True
```

Although valid Python, it may make the code less clear.

Prefer meaningful variables with consistent purposes.

---

14. Variables Store References to Values

A useful mental model is to think of a variable as a name referring to an object.

```
age = 25
```

Conceptually:

```
age ———→ 25
          int
```

And:

```
name = "Hafsa"
```

Conceptually:

```
name ———→ "Hafsa"
           str
```

The variable name itself does not permanently have one type.

The value it currently refers to has a type.

---

15. Data Type Determines Available Operations

Different types support different operations.

For example, strings support text operations:

```
name = "Hafsa"
```

```
print(name.upper())
```

Output:

```
HAFSA
```

Integers support mathematical operations:

```
age = 25

print(age + 5)
```

Output:

```
30
```

Booleans are useful for logical decisions:

```
is_active = True

if is_active:
    print("Account is active")
```

Understanding the type helps you understand what Python can do with the value.

---

16. Combining Different Data Types

Python sometimes allows different numeric types to work together.

```
age = 25
bonus = 2.5

result = age + bonus

print(result)
```

Output:

```
27.5
```

The integer participates in the calculation with the float, and the result is a float.

But Python does not automatically combine arbitrary types.

For example:

```
name = "Hafsa"
age = 25
```

```
print(name + age)
```

This causes:

```
TypeError
```

You must convert the value when appropriate.

---

17. Type Conversion

You can convert values between compatible types.

Common conversion functions include:

```
int()
float()
str()
bool()
```

For example:

```
age_text = "25"

age = int(age_text)

print(age)
print(type(age))
```

Output:

```
25
<class 'int'>
```

17.1 Integer to Float

```
price = 100

price = float(price)

print(price)
```

Output:

```
100.0
```

17.2 Number to String

```
age = 25

message = "I am " + str(age) + " years old."

print(message)
```

Output:

```
I am 25 years old.
```

Type conversion will be covered in depth in the next PDF.

---

18. Data Types and User Input

One of the most important things beginners should understand is that `input()` returns a string.

Consider:

```
age = input("Enter your age: ")
```

Even if the user enters:

```
25
```

Python receives:

```
"25"
```

So:

```
print(type(age))
```

produces:

```
<class 'str'>
```

If you want an integer:

```
age = int(input("Enter your age: "))
```

Now the result is an integer.

This distinction is essential when building interactive programs.

---

19. Data Types in a haas.dev Example

Imagine a haas.dev learning dashboard stores basic learner information.

```
name = "Hafsa"
age = 25
course_progress = 87.5
has_completed_python = False
certificate = None
```

Each value has a different purpose.

```
name                → str
age                 → int
course_progress     → float
has_completed_python→ bool
certificate         → NoneType
```

The program can then use those values differently.

```
if has_completed_python:
    certificate = "Python Certificate"
```

This is how data types become part of real application logic.

---

20. Data Types in a Web Application

Consider a product dashboard.

```
product_name = "Python Course"
price = 49.99
students = 1250
is_available = True
discount = None
```

The application needs different types because the information has different meanings.

```
product_name → text
price        → decimal number
students     → whole number
is_available → true/false
discount     → currently no value
```

Trying to represent everything as strings would make calculations and decisions harder.

---

21. Common Beginner Mistakes

Mistake 1: Confusing `"25"` with `25`

```
age = "25"
```

This is a string.

```
age = 25
```

This is an integer.

Mistake 2: Assuming input() Returns Numbers

```
age = input("Age: ")

print(age + 5)
```

This causes an error because `age` is a string.

Use:

```
age = int(input("Age: "))
```

Mistake 3: Mixing Strings and Numbers

Incorrect:

```
age = 25

print("Age: " + age)
```

Correct:

```
print("Age: " + str(age))
```

Or, more commonly:

```
print(f"Age: {age}")
```

Mistake 4: Confusing None with False

```
result = None
```

does not mean:

```
result = False
```

They represent different concepts.

Mistake 5: Using the Wrong Type for the Job

Suppose you need to store whether an account is active.

Using:

```
is_active = "yes"
```

works as text, but a boolean is usually more appropriate:

```
is_active = True
```

Choose types according to the meaning of the data.

---

22. How to Choose the Right Basic Type

Ask:

Is it text?

Use:

```
str
```

Example:

```
username = "Hafsa"
```

Is it a whole number?

Use:

```
int
```

Example:

```
students = 100
```

Is it a decimal number?

Use:

```
float
```

Example:

```
rating = 4.7
```

Is it true or false?

Use:

```
bool
```

Example:

```
is_verified = True
```

Is there currently no value?

Use:

```
None
```

Example:

```
profile_image = None
```

---

23. Checking Multiple Values

You can inspect several variables together.

```
name = "Hafsa"
age = 25
progress = 87.5
active = True
certificate = None

print(type(name))
print(type(age))
print(type(progress))
print(type(active))
print(type(certificate))
```

This is useful while learning and debugging.

---

24. A Practical Example

Let's build a simple learner status program.

```
name = "Hafsa"
age = 25
progress = 87.5
is_active = True

print("Learner:", name)
print("Age:", age)
print("Progress:", progress)
print("Active:", is_active)
```

Output:

```
Learner: Hafsa
Age: 25
Progress: 87.5
Active: True
```

Now use the values in logic:

```
if is_active and progress >= 80:
    print("Learner is making strong progress.")
```

The program works because each piece of information has an appropriate type.

---

25. What Happens When Types Don't Match?

Python raises errors when an operation does not make sense for the given types.

Example:

```
name = "Hafsa"
age = 25

result = name + age
```

Python raises:

```
TypeError
```

The important lesson is not to memorize every error message.

Instead, learn to ask:

> "What types are involved in this operation?"

You can inspect them:

```
print(type(name))
print(type(age))
```

This simple habit makes debugging much easier.

---

26. Type Awareness as a Debugging Skill

Suppose this code behaves unexpectedly:

```
price = input("Enter price: ")
quantity = input("Enter quantity: ")
```

```
total = price * quantity
```

The first debugging question should be:

```
print(type(price))
print(type(quantity))
```

You will discover that both are strings.

Convert them:

```
price = float(input("Enter price: "))
quantity = int(input("Enter quantity: "))

total = price * quantity

print(total)
```

Understanding types allows you to identify the actual problem instead of randomly changing code.

---

27. Basic Type Cheat Sheet

```
str
    Text
    "Hafsa"

int
    Whole numbers
    25

float
    Decimal numbers
    25.5

bool
    True or False
    True

NoneType
    No value
    None
```

---

28. Quick Comparison

Value	Type	Example
`"Hafsa"`	`str`	Text
`25`	`int`	Whole number
`25.5`	`float`	Decimal number
`True`	`bool`	Logical value
`None`	`NoneType`	No value

---

29. Mini Quiz

Question 1

What is the type of this value?

```
"100"
```

A. `int`

B. `float`

C. `str`

D. `bool`

Answer: C — `str`

Question 2

What is the type of:

```
75.5
```

A. `int`

B. `float`

C. `str`

D. `bool`

Answer: B — `float`

Question 3

What type does `input()` normally return?

A. `int`

B. `float`

C. `str`

D. `bool`

Answer: C — `str`

Question 4

What is the type of:

```
None
```

A. `bool`

B. `NoneType`

C. `str`

D. `int`

Answer: B — `NoneType`

Question 5

What does this produce?

```
type(25)
```

A. `str`

B. `float`

C. `int`

D. `bool`

Answer: C — `int`

---

30. 5 Minute Challenge

Create a Python program that stores information about a learner.

Your program should contain:

- a name
- an age
- a course progress percentage
- an account status
- a certificate value

Use the correct data type for each.

For example, your variables should represent concepts similar to:

```
name = ...
age = ...
progress = ...
is_active = ...
certificate = ...
```

Then print the type of every variable.

Expected structure:

```
Name type: ...
Age type: ...
Progress type: ...
Active type: ...
Certificate type: ...
```

Bonus Challenge

Create a variable containing:

```
"25"
```

Convert it into an integer and add `10`.

Expected result:

```
35
```

---

31. Practice Exercises

Easy

1. Create a string containing your name.
2. Create an integer containing your age.
3. Create a float containing a price.

4. Create a boolean indicating whether a user is logged in.
5. Create a variable containing `None`.
6. Print the type of each value.

Medium

1. Create five variables representing a course learner.
2. Use `type()` to inspect each variable.
3. Create a string containing a number.
4. Convert it to an integer.
5. Perform a mathematical calculation with the converted value.
6. Create a program that accepts an age using `input()` and converts it to an integer.

Hard

Build a small learner information program.

It should:

1. Ask for the learner's name.
2. Ask for their age.
3. Ask for their course progress.
4. Convert numerical inputs into appropriate types.
5. Store whether the learner is active.
6. Print all values.
7. Print the type of every value.
8. Display a message if progress is greater than or equal to 80%.

---

32. Interview Questions

1. What is a data type?

A data type defines the kind of value stored or referenced by a Python object and determines what operations can be performed with it.

2. What are common basic Python data types?

Common basic types include:

```
str
int
float
bool
NoneType
```

3. What is the difference between `10` and `10.0`?

```
10
```

is an `int`.

```
10.0
```

is a `float`.

4. What does `type()` do?

It returns the type of an object.

```
type(25)
```

returns an integer type.

5. What does `input()` return?

`input()` returns a string.

If numerical data is required, conversion is necessary.

6. Is Python statically typed or dynamically typed?

Python is dynamically typed. Variables do not require explicit type declarations before assignment.

7. What is `None`?

`None` represents the absence of a value and belongs to the `NoneType` type.

8. Why does this fail?

```
"25" + 5
```

Because the values are different types: `str` and `int`.

---

33. Key Takeaways

Remember these principles:

1. **Every Python value has a type.**
2. **Strings store text.**
3. **Integers store whole numbers.**
4. **Floats store decimal numbers.**
5. **Booleans represent `True` or `False`.**
6. **`None` represents the absence of a value.**
7. **Use `type()` to inspect a value's type.**

8. **`input()` returns a string.**
9. **Different types support different operations.**
10. **Convert values when the required operation needs another type.**
11. **Python is dynamically typed.**
12. **Choosing the correct type makes programs easier to reason about and debug.**

---

34. Final Mental Model

When you see a value in Python, ask three questions:

1. What kind of value is this?

```
25
```

→ `int`

2. What can I do with it?

```
25 + 10
```

→ mathematical operation

3. Do I need to convert it?

```
"25"
```

If you need it for mathematical calculations:

```
int("25")
```

This simple mental model will become increasingly important as you learn lists, dictionaries, functions, classes, APIs, databases, and real Python applications.

---

haas.dev

https://dev-roast-app.vercel.app

Visit haas.dev for more resources and guides.