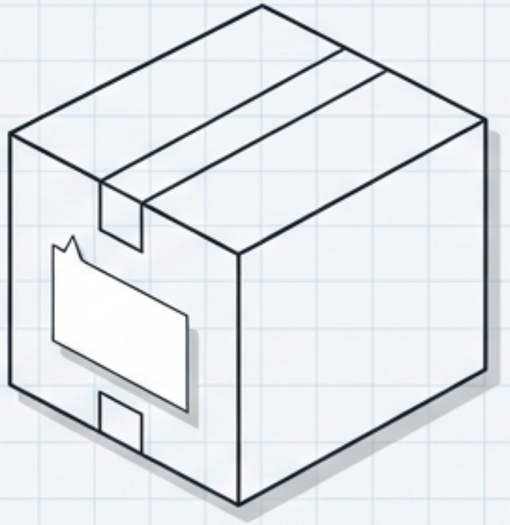
Ne ue Haas Grotesk Display Pro The Core Components of Python

A Blueprint for Variables and Data Types

The Fundamental Building Block: The Variable

In Python, a variable is a named container used to store data in memory. Think of it as a labelled box where you can keep a piece of information.

Python is dynamically typed, meaning you do not need to declare the type of data a variable will hold. The interpreter automatically detects the type when you assign a value.



Key Features

- **Dynamic Typing**: The type is determined at runtime.
- **Automatic Memory Allocation**: Memory is assigned when a value is given.
- **Flexible**: The same variable can hold different data types later on.

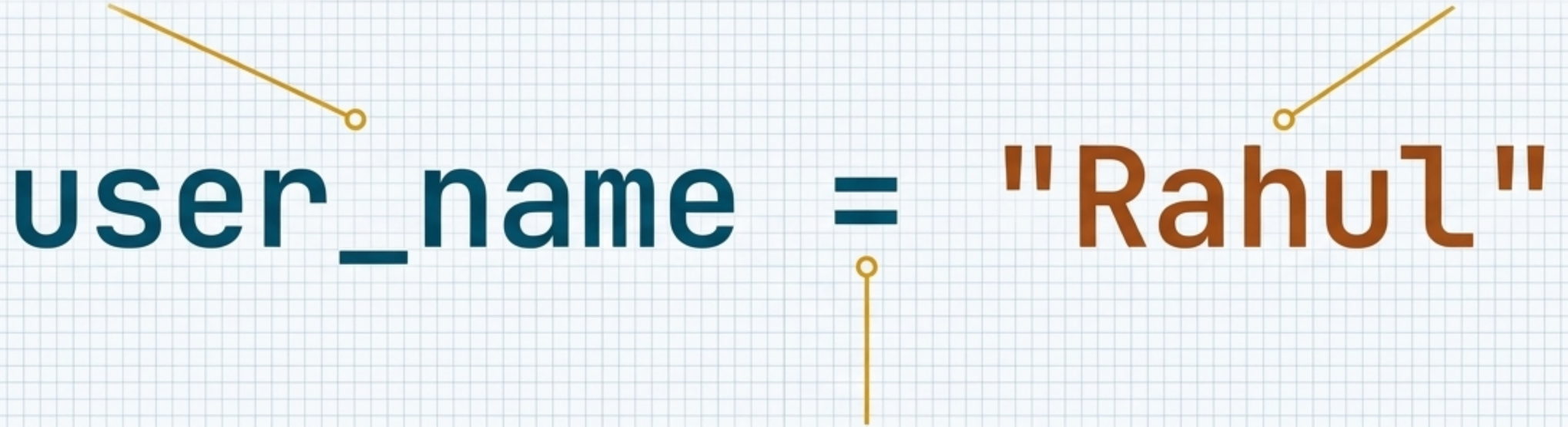
```
# 'x' now holds an integer
x = 10

# 'name' now holds a string
name = "Yaswant"
```

Anatomy of a Variable Assignment

Variable Name
(The label on our container)

Value / String Literal
(The data being stored)



Assignment Operator
(The action of 'placing into')

The Rules of Construction: How to Name Variables

To ensure your code is valid and readable, variable names must follow a specific set of rules.

aZ 9_

Allowed Characters:: May contain letters (a-z, A-Z), numbers (0-9), and the underscore character (`_`).

Starting Character:: Must begin with a letter or an underscore. It cannot start with a number.

a≠A

Case-Sensitivity:: Python distinguishes between uppercase and lowercase letters. `age` and `Age` are two different variables.

Reserved Keywords:: You cannot use Python's built-in keywords (like `if`, `for`, `class`) as variable names.

Naming Conventions in Practice

Valid Inter

```
# A simple, descriptive name
age = 20
```

Valid Name

```
# Using an underscore to separate words
user_name = "Rahul"
```

```
# Names can include numbers
price1 = 99
```

Invalid Inter

```
# ERROR: Cannot start with a number
1age = 20
```

ERROR: Starts with a number

```
# ERROR: Hyphens are not allowed
user-name = "A"
```

ERROR: Hyphen not allowed

Defining Unchanging Values: The Constant Convention

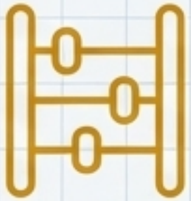
Python does not have built-in constants in the same way other languages do. However, there is a strong and widely followed convention: variable names written **entirely in uppercase** are **treated as constants** and should not be changed.

```
# A mathematical constant
PI = 3.14

# A configuration setting
MAX_SPEED = 120
```

The Raw Materials: An Overview of Python's Data Types

A variable can hold many different kinds of data. These are called data types. Understanding these core types is essential for writing any program. We will focus on the most common ones.



`int`

Integers

`float`

Floating-Point Numbers

`str`

Strings

`bool`

Booleans

`NoneType`

The None Value

Python also has more complex data structures like `list`, `tuple`, `dict`, and `set` for grouping data, as well as `complex` for mathematical applications.

Material Focus: Numeric Types

Integer (int)

Represents whole numbers, both positive and negative, without any decimal part.

```
x = 10
y = -5
```

Float (float)

Represents numbers that have a decimal part. Used for fractional values and precision measurements.

```
pi = 3.14
rate = 0.95
```

Material Focus: Textual Data (`str`)

The string type is used to store text data. In Python, you can create strings using single (`'`) or double (`"`) quotes.

```
# A simple greeting message
msg = "Hello"
```

Special Case: Multi-line Strings

For text that spans multiple lines, you can use triple quotes (`\"\"\"` or `'''`).

```
documentation = """This is a multiline string.
                It is often used for docstrings and
                long-form text blocks."""
```

Material Focus: Logic and Absence

Boolean (bool)

A boolean value can only be one of two things: True or False. It is the foundation of logic and control flow in programming.

```
flag = True
is_valid = False
```

The None Type (NoneType)

This special type has only one possible value: None. It is used to signify the absence of a value or an empty state.

```
# 'data' currently holds nothing
data = None
```

The Inspector's Tool: Verifying Data with type()

How can you check what type of data a variable holds? Python provides a built-in function called `type()` for exactly this purpose. It returns the type object of its argument.

```
# Checking an integer
type(10)
```

```
# Output: <class 'int'>
```

```
# Checking a string
type("Hello")
```

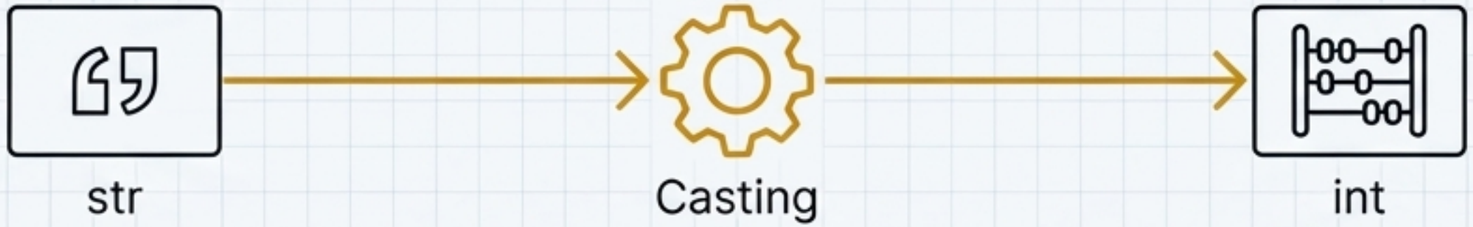
```
# Output: <class 'str'>
```

```
# Checking a boolean
flag = True
type(flag)
```

```
# Output: <class 'bool'>
```

The Transformer: Changing a Variable's Type

Sometimes you need to convert data from one type to another. This process is called type casting. Python provides simple, built-in functions for these conversions.



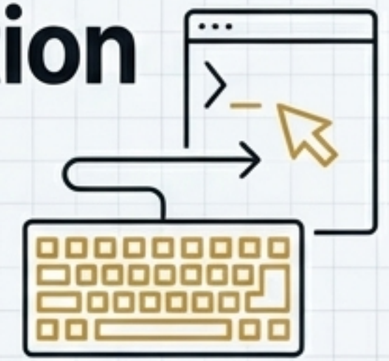
`int(), float(), str(), bool()`

```
# Casting a string to an integer
x = int("10")
# x is now the number 10

# Casting a string to a float
y = float("5.6")
# y is now the number 5.6

# Casting a number to a string
z = str(100)
# z is now the text "100"
```

Working with Live Data: The `input()` Function



To make your programs interactive, you can get input directly from the user with the `input()` function. It displays a prompt and waits for the user to type something and press Enter.

The Critical Rule of `input()`

The `input()` function **always** returns the user's data as a **string** (`str`), even if they type in numbers.

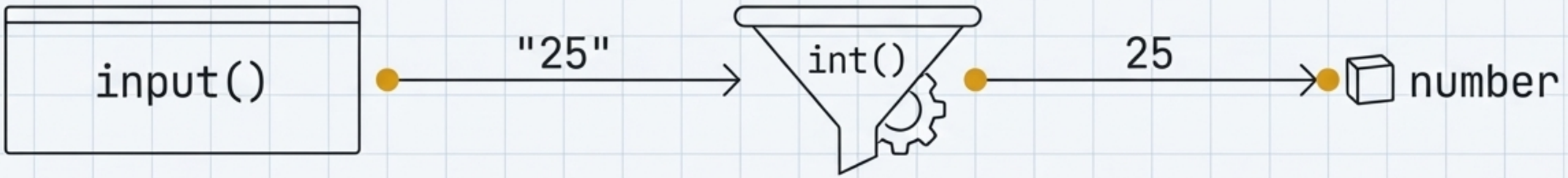
```
name = input("Enter your name: ")
# 'name' will be a string, e.g., "Rahul"

age_as_string = input("Enter your age: ")
# If the user types 25, 'age_as_string' will be "25", not the number 25
```

Neue Haas Grotesk Display Pro: "Blueprint for Interaction: Getting Numeric Input"

To get a number from the user, you must combine two tools: `input()` to get the text, and a **type casting function** like `int()` or `float()` to convert that text into a number.

The Pattern




```
# Prompt the user and immediately convert the string result to an integer
number = int(input("Enter a number: "))

# Now you can perform mathematical operations with 'number'
doubled = number * 2
print(doubled)
```

Workshop Efficiencies: Advanced Assignment and Cleanup

As you become more proficient, you can use these Python features to write more concise and readable code.

Multiple Assignments

Assign different values to multiple variables in a single line.

```
a, b, c = 10, 20, 30
```

Chained Assignments

Assign the same value to multiple variables at once.

```
x = y = z = 5
```

Removing a Variable

If you no longer need a variable, you can remove it from memory using the `del` keyword.

```
del x
```