Getting User Input & Type Conversion

AiCE Software Engineering Foundations — Python Programming

Every useful program communicates with the people who use it. Until now, your programs have worked only with data written directly into the code. This lesson changes that — you will learn to ask questions, receive answers, and respond to whatever the user provides.

1. What Is User Input?

User input is information a person provides to a running program while it's executing. Instead of every value being written into the code beforehand, the program pauses, displays a prompt, waits for the user to type something and press Enter, and then uses what they typed to continue.

This is the difference between a static program and an interactive one. A static program always does the same thing — it runs, produces the same output, and exits. An interactive program responds differently depending on who's using it and what they provide. Login forms, search bars, registration pages, banking apps — virtually every real application you've used waits for input and responds to it.

In Python, user input is handled with a single built-in function: input().

2. The input() Function

input() does two things: it displays a message — called a prompt — to the user, and it waits for the user to type something and press Enter. Whatever they type is returned as a string and can be stored in a variable.

```
variable = input("Your prompt message here: ")
```

Your first interactive program:

```
name = input("What is your name? ")
print(f"Welcome to AiCE, {name}!")
```

Running this displays "What is your name?" and waits. Typing Amara and pressing Enter produces "Welcome to AiCE, Amara!" — the program asked a question, waited for the answer, and used it in its response. This is the foundation of every interactive application ever built.

> **The Space After the Colon**
> *Notice the trailing space in "What is your name? " — that space is intentional. Without it, the cursor appears immediately after the colon with no visual gap, making the prompt feel cramped. Always include a space at the end of your input prompts.*

3. input() Always Returns a String

This is the single most important rule about input(), and the source of many beginner errors: no matter what the user types — a name, a number, a date — input() always returns it as a string.

```
age = input("Enter your age: ")
print(f"You entered: {age}")
print(f"Data type : {type(age)}")
```

Typing 22 when prompted produces:

```
You entered: 22
Data type : <class 'str'>
```

Even though you typed a number, Python stored it as the string "22". Trying arithmetic with it directly — age + 1 — would raise a TypeError, since you can't add an integer to a string. The fix is type conversion.

4. Type Conversion — int(), float(), and str()

Type conversion changes data from one type to another. Python provides three functions for the most common conversions:

Function	Converts To	Example	Result
int(value)	Integer	int("25")	25
float(value)	Float	float("3.14")	3.14
str(value)	String	str(100)	"100"

Converting input to an integer so it can be used in arithmetic:

```
age = int(input("Enter your age: "))
birth_year = 2026 - age
print(f"You were born around {birth_year}.")
```

> **Read From the Inside Out**
> *age = int(input("Enter your age: ")) runs the inner function first — input() collects the text — then passes it immediately to the outer function, int(), which converts it. This convert-and-assign-in-one-line style is the more common professional pattern, compared to converting on a separate line afterward.*

Converting input to a float, for decimals like a temperature or a price:

```
celsius = float(input("Enter temperature in Celsius: "))
fahrenheit = (celsius * 9/5) + 32
print(f"{celsius}°C is equal to {fahrenheit:.1f}°F")
```

Entering 38.5 produces: 38.5°C is equal to 101.3°F.

5. What Happens When the User Enters the Wrong Type

When int() or float() tries to convert text that genuinely can't be converted — typing 'twenty-two' when a number was expected — Python raises a ValueError and the program crashes:

```
Enter your age: twenty-two
ValueError: invalid literal for int() with base 10:
'twenty-two'
```

Handling This Properly Comes Later

A later lesson on exception handling teaches you to catch errors like this and respond gracefully instead of crashing. For now, in these beginner programs, we assume users enter the correct type of data — but it's worth knowing exactly why this error happens when it does.

What you can do right now is write clear, specific prompts that reduce the chance of a user entering the wrong thing:

Vague Prompt	Clear Prompt
Enter a value:	Enter your age as a whole number (e.g. 22):
Price:	Enter the price in Naira (e.g. 5000.50):

User Input Is Untrusted

In professional programming, user input is always treated as untrusted — you never assume users will enter exactly what you expect. Validation, cleaning, and exception handling are how programs become robust; these techniques are introduced progressively as this program continues.

Key Takeaways

input() always returns a string, no matter what was typed — which means numbers need explicit conversion with int() or float() before any arithmetic can happen on them. The convert-and-assign-in-one-line style, like age = int(input(...)), is the professional standard. A clear, specific prompt reduces user error more than any amount of code ever will. The next lesson puts this into practice, building genuinely interactive programs — calculators, converters, and a complete registration system — that combine input() with everything covered in previous lessons.