مكتب

Quick Reference Notes

Week 4: Making Programs Interactive

7th Grade Computer Science

1 Introduction

1.1 This Week's Big Question

How can we make a computer interact with us? Until now, our programs have been one-way conversations - Python talking to us through `print()`. This week, we'll learn how to create programs that listen, understand, and respond to us!

Prerequisites

Before starting this week, you should be comfortable with:

- Creating and using variables, familiarity with `int`, `float`, `str`, `bool`
- Using the `print()` function
- Understanding that Python reads code top to bottom
- Knowing the difference between assignment (`=`) and equality (`==`)

1.2 What You Already Know

In Scratch, you used the "ask and wait" block to get information from users. You've also created variables to store different types of data. Now we'll combine these skills to make interactive Python programs!

1.3 What You'll Be Able to Do

By the end of this week, you'll:

- Get information from users with the `input()` function
- Convert text input into numbers for calculations
- Format output to make it look professional

2 VIDEO 1: The input() Command - Python Listens!

2.1 Making Programs Interactive

So far, our programs have been like TV shows - they display information but can't listen to you. The `input()` command changes that! It's like giving your program ears to listen. It waits for the user to type something.

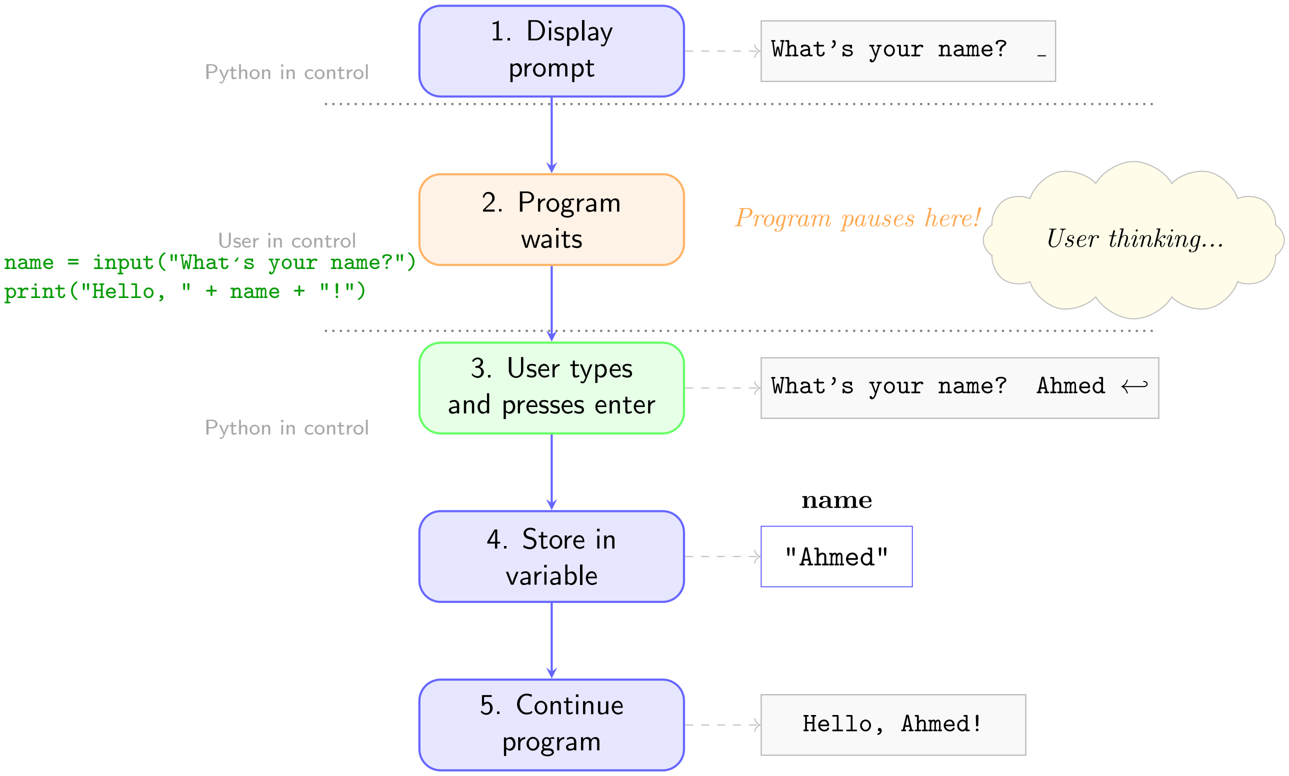
2.2 Let's See It In Action

```python
name = input("What's your name? ")
print("Hello, " + name + "!")
print("Nice to meet you.")
```

When you run this:

1. Python displays: `What's your name?`
2. The program waits for you to type your response
3. You type: `Ahmed` and press Enter
4. Python stores `"Ahmed"` in the variable `name`
5. Then continues with the rest of the program

How `input()` Works



```python
name = input("What's your name?")
print("Hello, " + name + "!")
```

```
What's your name? _
What's your name?  Ahmed ↩
"Ahmed"
Hello, Ahmed!
```

2.3 Breaking Down input()

The `input()` function has two jobs:

- **Display a prompt** - Shows text to guide the user
- **Wait and return** - Waits for `Enter` key, then returns what was typed

Rule

input() Pattern:

`variable = input("prompt text")`

- The prompt appears exactly as written
- Add a space at the end for better formatting, for example: `input("Enter your name: ")`
- Whatever the user types becomes a string

Landmark Moment

This is an important moment! Your programs can now have conversations with users. You've transformed from creating static programs to building interactive applications. This is the same principle used in every app you use - from WhatsApp to games!

2.4 input() Always Returns a String

Here's something critical to remember:

```python
age = input("How old are you? ")
```

If user types 13, age now contains the string `"13"`, not the number 13

```python
print(type(age))  # Shows: <class 'str'>
```

Common Error

input() always returns a string, even if the user types numbers!

This will crash:

```python
age = input("Enter your age: ")  # User types: 13
next_year = age + 1              # ERROR! Can't add string + number
```

Why? Because age contains `"13"` (text), not 13 (number)

Tip

Good prompts help users know what to type:

- Be specific: `"Enter your age in years:"`, not just `"Age:"`
- Include units: `"Enter temperature in Celsius:"`
- Show format: `"Enter date (DD/MM/YYYY):"`

Quick Check

What will this program display if the user types `Fatima`?

```python
student = input("Enter student name: ")
print("Welcome")
print(student)
```

3 VIDEO 2: Type Conversion - Making Numbers from Strings

3.1 What Are Functions?

Before we dive into type conversion, let's understand a new term. In Python, when you see something with parentheses like `print()`, `input()`, or `int()`, these are called **functions**.

Think of functions as Python's built-in tools:

- `print()` is a tool that displays things on screen
- `input()` is a tool that gets text from the user
- Some tools can transform data from one type to another!

The parentheses `()` are where you put information for the tool to work with - like putting material into a machine.

3.2 The Type Conversion Problem

Remember from last video that `input()` always gives us strings? But what if we need numbers for math? We can't add `"5" + "3"` to get 8 - we'd get `"53"` instead!

This is where Python's type conversion tools come to the rescue. They transform strings into numbers (and vice versa) so we can do calculations.

3.3 Meet the Type Converters

Python has three main type conversion functions:

- `int()` - converts to integers (whole numbers)
- `float()` - converts to floats (decimal numbers)
- `str()` - converts to strings (text)

3.4 Converting User Input to Numbers

Here's the pattern for getting numbers from users:

```python
age_text = input("How old are you? ")      # Get 13 as string "13"
age = int(age_text)                        # Convert to number 13
next_year = age + 1                        # Now we can do math!
print("Next year you'll be", next_year)    # Shows: 14

# One-line approach (more common):
age = int(input("How old are you? "))     # Get input and convert in one step
```

ℹ Tip

The Magic Pattern: Wrap `input()` inside a converter!

- `int(input(...))` - for whole numbers
- `float(input(...))` - for decimal numbers

3.5 How Each Converter Works

`int()` - Makes integers (chops off decimals!)

```python
score = int("100")        # "100" -> 100
count = int("42")         # "42"  -> 42
rounded = int(3.9)        # 3.9   -> 3 (decimal part removed!)
```

`float()` - Makes decimal numbers

```python
price = float("19.99")  # "19.99" -> 19.99
height = float("5.6")   # "5.6"   -> 5.6
whole = float("7")      # "7"     -> 7.0 (adds .0)
```

Rule

Remember:

- `int()` doesn't round - it chops! `int(9.9)` becomes 9, not 10
- `float()` keeps decimals and can handle whole numbers too

3.6 When Conversions Go Wrong

Not every conversion makes sense. Python will show an error if you try impossible conversions:

Common Error

These conversions will crash your program:

```python
int("hello")          # X Can't turn words into numbers!
int("12.5")           # X int() can't handle decimal strings
int("50 rupees")      # X Extra text confuses Python
float("ten")          # X Python doesn't understand word numbers
```

But these work perfectly:

```python
int("42")             # Clean whole number string
float("3.14")         # Clean decimal string
int(3.14)             # Converts float to int (becomes 3)
str(anything)         # str() can convert almost anything!
```

3.7 Putting It All Together

Let's build a program that uses multiple conversions:

```python
# A program to calculate your age in months
print("Age Calculator")
print("-" * 20)

name = input("What's your name? ")                # No conversion needed
years = int(input("How old are you in years? "))  # Convert to int
months = years * 12                               # Calculate

# Display results
print("\nHello", name + "!")
print("You are", years, "years old")
print("That's", months, "months!")
```

Tip

Which converter should I use?

- `int()` for counting: people, items, scores, age in years
- `float()` for measuring: height, weight, temperature, money
- When unsure, `float()` is safer - it accepts more number formats!

Quick Check

What happens in each case? (Try to predict before testing!)

1. `int("25")` → __________
2. `float("3.5")` → __________
3. `int(7.8)` → __________
4. `str(100)` → __________
5. `int("5.0")` → __________ (Careful!)

Quick Check

Fix this broken code:

```python
# This code has errors!
age = input("Your age: ")
double_age = age * 2
print("Double your age is", double_age)
```

Hint: What type does `input()` return?

4 VIDEO 3: String Formatting - Making Output Look Good

4.1 Why String Formatting Matters

When we combine text and numbers in output, we want it to look professional and be easy to read. String formatting gives us control over how our output appears.

4.2 Method 1: Comma Separation in print()

The simplest way - use commas to separate items:

```python
name = "Ali"
score = 95
print("Player:", name, "Score:", score)
# Output: Player: Ali Score: 95
```

Python automatically adds spaces between items

4.3 Method 2: String Concatenation with +

Join strings together with the + operator:

```python
first = "Ahmed"
last = "Khan"
```

```python
full_name = first + " " + last
print("Welcome, " + full_name + "!")
# Output: Welcome, Ahmed Khan!
```

Remember: The concatenation only works with strings. Combining text and numbers directly will cause an error:

```python
age = 13
print("Age: " + age)  # ERROR! Can't add string + number
print("Age: " + str(age))  # Convert to string first
```

4.4 Method 3: Format Strings (f-strings)

The modern, powerful way - put `f` before the string. This allows you to insert variables and expressions directly into the string by using curly braces {}:

```python
name = "Sara"
score = 88
lives = 3

# Put variables inside {} in an f-string
message = f"Player {name} has {score} points and {lives} lives"
print(message)
# Output: Player Sara has 88 points and 3 lives

# Can do calculations inside {}
print(f"Next year {name} will have {score + 10} points")
```

Rule

f-string Pattern:

`f"text {variable} more text {expression}"`

- Start with `f` before the quotes
- Put variables/expressions inside curly braces {}
- No need to convert numbers to strings. The `f`-string does it automatically!

4.5 Formatting Examples

Here are practical examples of each method:

```python
# Getting user information
name = input("Enter your name: ")
age = int(input("Enter your age: "))
height = float(input("Enter height in meters: "))

# Method 1: Commas (simple but less control)
print("Name:", name, "Age:", age, "Height:", height)

# Method 2: Concatenation (more control, more work)
print("Hello " + name + "! You are " + str(age) + " years old.")

# Method 3: f-strings (best of both worlds!)
print(f"Hello {name}! You are {age} years old and {height}m tall.")
```

Tip

Which method to use?

- Quick output: Use commas in `print()`
- Building strings: Use `f`-strings
- Rare (only when necessary): Use the + concatenation

Most Python programmers prefer `f`-strings for their clarity and power!

Quick Check

What will this `f`-string display?

```python
item = "notebook"
price = 45
quantity = 3
print(f"Buying {quantity} {item}s costs {price * quantity} rupees")
```

5 VIDEO 4: Summary

5.1 Making Programs Interactive

This week, we transformed our programs from one-way displays to interactive conversations:

- `input()` lets programs wait for and receive user input
- Type conversion functions `int()`, `float()` turn string input into numbers and `str()` converts numbers to strings
- String formatting makes our output professional and readable
- We can now create programs that adapt to user input

5.2 Key Syntax Patterns

The patterns we mastered this week:

```python
# Getting input
name = input("prompt text")

# Getting numbers from users
age = int(input("Enter age: "))
price = float(input("Enter price: "))

# String formatting
print("Text", variable, "more text")          # Commas
print("Text " + str(number))                  # Concatenation
print(f"Text {variable} and {expression}")    # f-strings
```

6 Quick Reference

Quick Reference

Week 4 Patterns:

- Get text: `variable = input("prompt")`
- Get integer: `number = int(input("prompt"))`
- Get decimal: `number = float(input("prompt"))`
- Convert to string: `str(number)`
- Format output: `f"text {variable} text"`
- Remember: `input()` always returns a string by default!

6.1 Reflection

Think about your learning this week:

1. What kind of interactive program would you like to create?
2. Why do you think `input()` always returns a string?
3. Which string formatting method do you find easiest to use?

Next week, we'll learn about expressions and operators to make our calculations more powerful!