مكتب

Quick Reference Notes

Week 3: Variables and Data Types

7th Grade Computer Science

1 Introduction

1.1 This Week's Big Question

How do computers store and process information? This week, we'll discover how Python remembers things using variables - the building blocks that turn simple print statements into powerful, interactive programs!

Prerequisites

Before starting this week, you should be comfortable with:

- Using the Thonny IDE (Editor and Shell)
- Writing and running `print` statements
- Understanding that Python reads code in order, from top to bottom

1.2 What You Already Know

In Scratch, you've created variables to keep score in games or remember a player's name. Python variables work the same way - they're containers that hold information your program needs to remember.

1.3 What You'll Be Able to Do

By the end of this week, you'll:

- Create variables to store different types of information
- Understand why you need to know if you're storing a number or text
- Use memory diagrams to visualize what's happening inside the computer
- Avoid common variable mistakes that confuse beginners

2 VIDEO 1: Variables - Python's Memory Boxes

2.1 What Are Variables?

Imagine you have a box where you can store one thing at a time. You put a label on it (like "`age`" or "`player_name`") so you remember what's inside. That's exactly what a variable is - a labeled box in

the computer's memory!

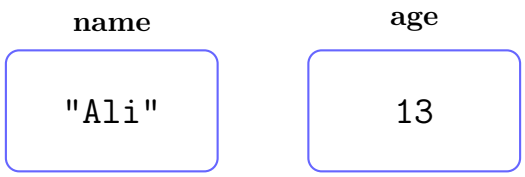
2.2 Creating Your First Variables

```
name = "Ali"
age = 13
print(name)
print(age)
```

2.3 Breaking It Down

When Python sees `name = "Ali"`, it:

1. Creates a box in memory
2. Labels it "`name`"
3. Puts "`Ali`" inside
4. Remembers this for later use



Rule

Variable Creation Pattern: `variable_name = value`

- Left side: The label for your box (variable name)
- Equal sign: Means "put into" (it's not compairing as in math!)
- Right side: What goes in the box

Common Error

Variables must be created before use!

```
# This will cause an error:
print(score)    # Error! 'score' doesn't exist yet
score = 100

# This works:
score = 100     # Create first
print(score)    # Now we can use it
```

Remember: Python reads top to bottom - create your variables before using them!

2.4 Choosing Good Variable Names

Just like labeling boxes in your room, good variable names help you (and others) understand your code!

Rule

Variable Naming Rules:

- Must start with a letter or underscore (_)
- Can contain letters, numbers, and underscores
- Cannot contain spaces or special characters
- Cannot be Python keywords (like `print`)
- Case matters! `name` and `Name` are different

```
# Good variable names - clear and descriptive
player_score = 100
student_age = 13
is_game_over = False
first_name = "Ahmed"

# Bad variable names - unclear or confusing
ps = 100          # What does 'ps' mean?
x = 13            # What does 'x' represent?
data = False      # What kind of data?
n = "Ahmed"       # Too vague!
```

Tip

Writing Great Variable Names:

- Use descriptive names: `player_health` not `ph`
- Use underscores for multiple words: `high_score` not `highscore`
- Be consistent: If you use `num_students`, don't switch to `studentCount`. Use same style.

Common Error

Variable names that will cause errors:

```
# These will ALL crash your program!
2nd_player = "Sara"      # Cannot start with number
player-name = "Ali"      # No hyphens allowed (Python thinks it's
    subtraction!)
my score = 95            # No spaces allowed
class = "7A"             # 'class' is a Python keyword
```

Python will immediately show an error and refuse to run your code!

Quick Check

Which variable names are valid in Python? (They might not be good names, but will they work?)

1. `Score`
2. `_score`
3. `score2`
4. `2score`
5. `my-score`
6. `my_score`

Quick Check

What will this code display?

```
city = "Lahore"
print(city)
```

3 VIDEO 2: The Four Types of Data

3.1 Why Data Types Matter

Python needs to know what kind of information you're storing because different types behave differently. It's like knowing whether a box contains a liquid or a solid - you handle them differently!

3.2 Meet the Four Basic Types

3.2.1 Integers (int) - Whole Numbers

These are whole numbers without any decimal point. They can be positive or negative, and they include zero.

```
students = 25
temperature = -5    # Negative numbers are fine!
year = 2024
```

3.2.2 Floats (float) - Decimal Numbers

These are numbers with a decimal point. They can be positive or negative, and they allow for fractions.

```
price = 49.99
height = 5.7
pi = 3.14159
```

3.2.3 3. Strings (str) - Text

Strings are sequences of characters, like words or sentences. They are always enclosed in quotes (single or double).

```
school = "Maktab"                # Double quotes
city = 'Lahore'                  # Single quotes work too!
message = "Welcome to Python!"
```

Tip

Python lets you use either single quotes (') or double quotes (") for strings - they work exactly the same! Pick one style and stick with it in your program.

Common Error

Don't mix quotes!

```
# These will cause errors:
name = "Ali'      # Started with " but ended with '
text = 'Hello"    # Started with ' but ended with "

# These work correctly:
name = "Ali"      # Both double quotes
name = 'Ali'      # Both single quotes
```

Always match your opening and closing quotes!

3.2.4 Booleans (bool) - True/False

Booleans represent truth values - they can only be `True` or `False`. They are often used for conditions in your code.

```
is_student = True
homework_done = False
is_raining = True
```

Tip

Quick way to remember strings: If it has quotes around it, it's a string - even `"123"` is text, not a number!

Quick Check

Identify the data type: What type is each of these?

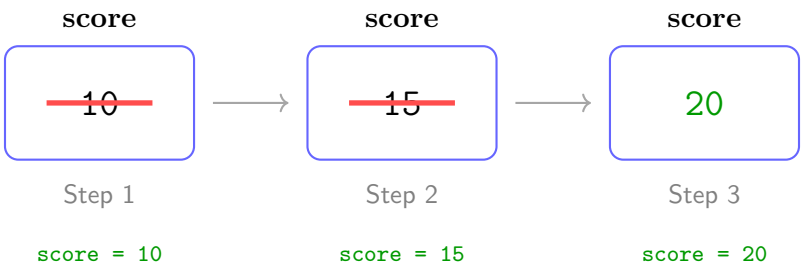
- `42`
- `"42"`
- `3.0`
- `True`

4 VIDEO 3: Variables Can Change!

4.1 The Power of Reassignment

Unlike a permanent marker, variables can be erased and rewritten. Watch what happens:

```
score = 10      # Step 1
print(score)    #    Shows: 10
score = 15      # Step 2
print(score)    #    Shows: 15
score = 20      # Step 3
print(score)    #    Shows: 20
```



Same variable, different values!

4.2 Variables Remember Only One Thing

When you put a new value in a variable, the old value disappears completely - like erasing a whiteboard before writing something new.

Common Error

Common Misconception: Variables do NOT store multiple values or remember their history. They only know their current value!

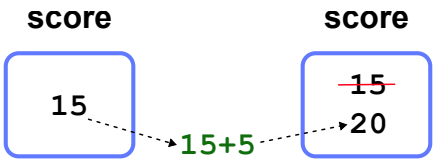
```
x = 5
x = 10
# x is now just 10.. Not 5 and 10 both!
```

4.3 Reading Values from Variables

When a variable appears on the right side of the equals sign Python reads its current value. For example, in the following code:

Python processes `score = score + 5` in three steps:

1. **Read**: Look at the right side first - what's the current value of 'score'? (15)
2. **Calculate**: Do the math: $15 + 5 = 20$
3. **Store**: Put the result (20) back into '`score`'



Tip

Think of it like updating your game score: You check your current score (read), add your new points (calculate), then write down the new total (store).

Quick Check

After this code runs, what is the value of `points` at the end of the program?

```
points = 100
points = 200
points = points + 50
```

4.4 String Addition - Joining Text Together

Now that you've seen how + works with numbers, here's something cool: you can also use + with strings! But instead of doing math, it joins (concatenates) the strings together.

```
# With numbers, + does math:
score = 10
score = score + 5      # 10 + 5 = 15

# With strings, + joins them:
message = "Hello"
message = message + " World"     # "Hello" + " World" = "Hello World"
print(message)     # Shows: Hello World

# Another example:
first = "Python"
second = "Programming"
combined = first + " " + second     # Note the space in between!
print(combined)     # Shows: Python Programming
```

Tip

Remember the difference:

- `2 + 3` gives you `5` (addition)
- `"2" + "3"` gives you `"23"` (joining)
- The quotes make all the difference!

Quick Check

What will this code display?

```
part1 = "Game"
part2 = "Over"
result = part1 + part2
print(result)
```

Did you remember to think about spaces?

5 VIDEO 4: Assignment vs. Equality

5.1 One Equal vs. Two Equals

This is where many beginners get confused! In Python:

- One equal sign (=) means "put into" or "assign"
- Two equal signs (==) means "is equal to? or comparing" *(we'll use this next week!)*

5.2 Assignment in Action

This is assignment, i.e putting values into variables:

```
age = 13            # Put 13 into age
name = "Sara"       # Put "Sara" into name
x = y               # Put y's value into x
```

Rule

Remember: One equal sign (=) "puts in", it does not "compare".

- `score = 100` means "put 100 into score"
- Read it right-to-left: "100 goes into score"

Common Error

These will cause errors:

- `13 = age` - Can't put age into 13!
- `"Hello" = message` - Can't put message into "Hello"!

Always: variable name on **left**, value on **right**

Quick Check

Which of these are valid assignments?

1. `color = "blue"`
2. `"red" = color`
3. `x = 5 + 3`
4. `10 = y`

6 VIDEO 5: Memory Diagrams - Seeing Inside the Computer

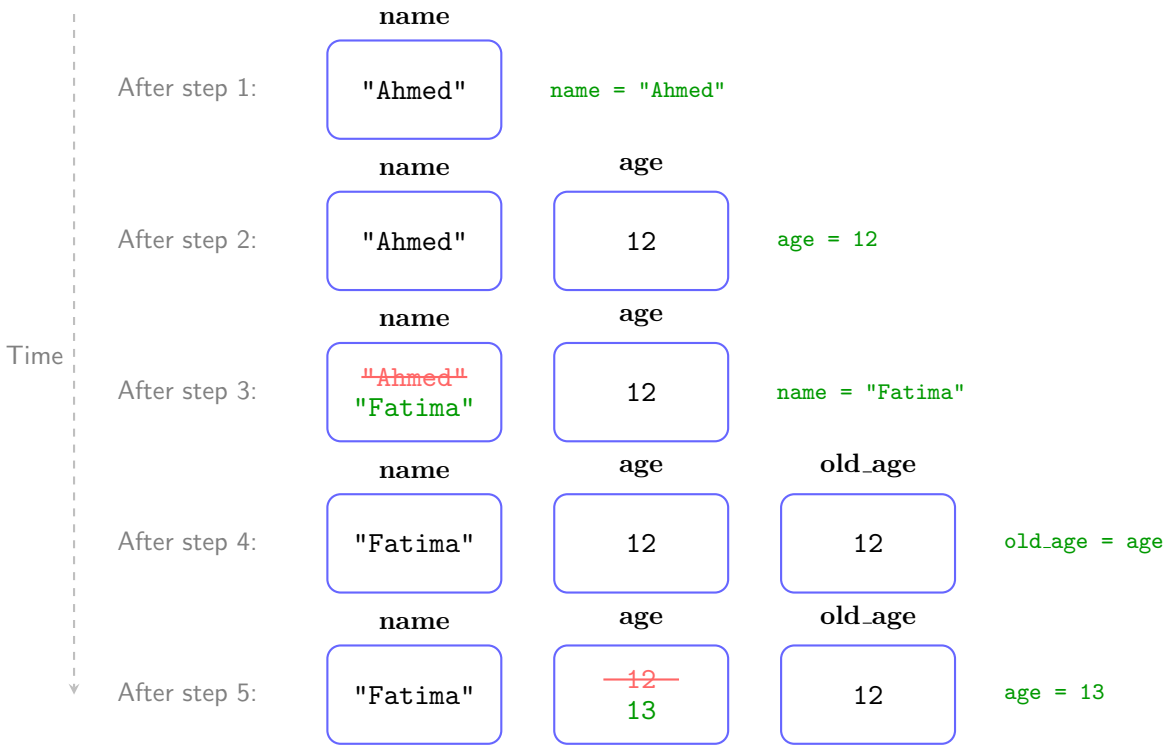
6.1 Visualizing Variables

Memory diagrams help us see what Python is doing behind the scenes. They're like X-ray vision for your code!

6.2 Drawing Memory Diagrams

Let's trace through this code step by step:

```
name = "Ahmed"        # Step 1
age = 12              # Step 2
name = "Fatima"       # Step 3
old_age = age         # Step 4
age = 13              # Step 5
```



6.3 Understanding the Memory Diagram

Let's decode what this diagram shows us:

Each row is a snapshot in time - like taking a photo of Python's memory after each line of code runs. Time flows from top to bottom, just like how Python reads your code!

Why are some boxes missing? Notice how:

- Step 1: Only `name` exists because that's all we've created
- Step 2: Now `age` appears because we just created it
- Step 3: `name` is still the same box, but with new contents
- Step 4: A third box `old_age` appears when we create it
- Step 5: Same three boxes, but `age` has new contents

Python doesn't know about `age` in Step 1 because the line `age = 12` hasn't run yet. This is why using a variable before creating it causes an error.

The red strikethrough shows when a value gets replaced. The box stays the same (same variable), but the contents change. It's like erasing and rewriting on the same whiteboard.

Rule

Variables don't exist until you create them. Variables are created when you assign a value to them for the very first time. If you try to use a variable that hasn't been created yet, Python will give you an error.

The second time you assign a value to a variable, it replaces the old value with the new one. The old value is gone forever - Python doesn't remember it!

Tip

When drawing memory diagrams:

- Draw a box for each variable
- Write the variable name above/beside the box
- Put the current value inside
- Cross out old values when they change

Quick Check

Draw a memory diagram after this code runs:

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

What are the final values of x and y?

7 VIDEO 6: Summary

7.1 Variables Are Python's Memory

This week, we learned how Python remembers information:

- Variables are labeled boxes that store one value at a time
- Python has four basic data types: `int`, `float`, `str`, and `bool`
- Variables can change their values (the old value disappears)
- The `+` operator works differently: math for numbers, joining for strings
- One equal sign (`=`) means assignment, not mathematical equality
- Memory diagrams help us visualize what's happening inside Python

7.2 Key Syntax Patterns

The patterns we mastered this week:

```
# Creating variables
variable_name = value

# Different data types
age = 13                # int
height = 5.5            # float
name = "Student"        # str
is_ready = True         # bool

# Changing variables
score = 10
score = score + 5       # Now score is 15

# Combining strings
greeting = "Hello" + " " + "World"  # Now greeting is "Hello World"
```

8 Quick Reference

Quick Reference

Week 3 Patterns:

- Create variable: `name = value`
- Integer: `age = 13`
- Float: `price = 9.99`
- String: `message = "Hello"`
- Boolean: `is_valid = True`
- Read and Change variable: `x = x + 1`
- Combine strings: `full = first + " " + last`
- Remember: = means "put into", not equals!

8.1 Reflection

Think about your learning this week:

1. Which data type seems most useful for the programs you want to create?
2. Draw a memory diagram for a simple program you write
3. Explain to a friend why `10 = x` doesn't work in Python
4. What happens when you use `+` with `"5" + "3"` versus `5 + 3`? Why is this important to remember?

Next week, we'll make our programs interactive by getting input from users and converting between different data types!