Python Programming Bootcamp
Week 1: Introduction to Programming & Python Fundamentals

Hilal Khan

Center for Artificial Intelligence & Emerging Technologies

What We'll Cover This Week

- Introduction to Programming - Understanding how computers work
- Why Python? - Exploring Python's advantages and applications
- Setting Up Python - Installation and development environment
- Your First Python Program - Hello World and basic syntax
- Variables and Data Types - Storing and working with data
- Python Operators - Arithmetic, comparison, and logical operators
- Type Conversion (Casting) - Converting between data types
- Getting User Input - Making programs interactive
- Basic Input/Output - Formatting and displaying information
- Conditional Statements - Making decisions in code
- Loops - For loops, while loops, and loop control
- Putting It All Together - Complete examples and practice

Introduction to Programming

What is Programming?

- **Programming** is the process of creating instructions for computers
- It's like writing a recipe - step by step instructions
- Computers are very literal - they do exactly what you tell them
- Programs solve problems and automate tasks

Real World Analogy
Think of programming like giving directions to a friend:

- Be specific and clear
- Break down complex tasks into simple steps
- Handle different scenarios (what if?)

How Computers Think

Humans:

- Intuitive
- Context-aware
- Can guess meaning
- Handle ambiguity

Computers:

- Literal
- Need explicit instructions
- No assumptions
- Very fast and accurate

Key Point
Programming is translating human problem-solving into computer language!

Programming Languages

- **High-level languages:** Python, Java, C++, JavaScript
- **Low-level languages:** Assembly, Machine Code
- Each language has its strengths and use cases
- We choose Python because it's:
  - Easy to read and write
  - Powerful and versatile
  - Great for beginners
  - Widely used in industry

Why Python?

Why Choose Python?

Python's Philosophy:

- "Beautiful is better than ugly"
- "Simple is better than complex"
- "Readability counts"
- "There should be one obvious way to do it"

Python is Used For:

- Web Development (Instagram, YouTube)
- Data Science & AI (Netflix, Google)
- Automation & Scripting
- Scientific Computing
- Game Development

Python vs Other Languages

Language	Hello World	Lines
Python	`print("Hello World")`	1
Java	`public class...`	5+
C++	`#include <iostream>...`	6+

Python Advantage
Python lets you focus on solving problems, not fighting with syntax!

Dynamic vs Static Typing i

Python (Dynamic Typing):

```
# No type declaration needed
x = 5          # x is int
x = "hello"    # x is now str
x = 3.14       # x is now float

# Type determined at runtime
print(type(x))  # <class 'float'>
```

C++ (Static Typing):

Dynamic vs Static Typing ii

```
// Type must be declared
int x = 5;           // x is always int
// x = "hello";      // ERROR!
// x = 3.14;         // ERROR!

string name = "John";  // Fixed type
float price = 19.99;   // Fixed type
```

Key Difference

Python: Variables can change type during execution

C++: Variables have fixed type determined at compile time

Interpreted vs Compiled Languages i

Python (Interpreted):

- Code executed line by line
- No compilation step needed
- Slower execution
- Easier debugging
- Cross-platform

```
$ python my_program.py
Hello World!
```

C++ (Compiled):

- Code converted to machine code
- Compilation step required

Interpreted vs Compiled Languages ii

- Faster execution
- Errors caught at compile time
- Platform-specific executables

```
$ g++ -o program main.cpp
$ ./program
Hello World!
```

Setting Up Python

Installing Python

Step-by-step installation:

1. Visit `https://www.python.org/downloads/`
2. Download Python 3.x (latest version recommended)
3. Run the installer
4. **Critical:** Check "Add Python to PATH"
5. **Critical:** Check "Install pip" (package manager)
6. Verify installation: Open terminal/command prompt
7. Type: `python -version` or `python3 -version`
8. Type: `pip -version`

Windows Users
If `python` doesn't work, try `python3` or `py`

Development Environment Options

Beginner-Friendly:

- **IDLE** - Comes with Python
- **Thonny** - Simple and clean
- **Online editors** - No installation needed

Professional:

- **VS Code** - Most popular
- **PyCharm** - Full-featured IDE
- **Sublime Text** - Lightweight

For This Course:

- We'll start with **IDLE**
- It's included with Python
- Perfect for learning
- No additional setup needed

Try Now

Search for "IDLE" in your start menu or applications

Introduction to IDLE i

IDLE (Integrated Development and Learning Environment)

Two Windows:

1. **Shell** - Interactive mode
2. **Editor** - Write scripts

Key Features:

- Syntax highlighting
- Auto-completion
- Built-in debugger
- Easy to use

IDLE Shell Window:

```
Python 3.11.0 (main, Oct 24
    2022)
[GCC 9.4.0] on linux
Type "help", "copyright"
   for more info.
>>> print("Hello World!")
Hello World!
>>> 2 + 3
5
>>>
```

Practice
Open IDLE and try these commands in the shell

Python Interactive Shell

Two ways to run Python:

1. **Interactive Mode** - Type commands directly
2. **Script Mode** - Write programs in files

Try This Now!
Open terminal/command prompt and type:

- `python`
- `>> print("Hello, World!")`
- `>> 2 + 3`
- `>> exit()`

Your First Python Program

Hello World Program i

Our first program:

```
print("Hello, World!")
```

Let's break it down:

- `print` is a **function**
- `()` contains the **arguments**
- `"Hello, World!"` is a **string**
- The function displays text on screen

Hello World Program ii

Practice
Try these variations:

```
print("Welcome to Python!")
print("My name is", "Hilal")
print(2023)
```

Python Syntax Rules i

Key syntax rules you MUST follow:

1. Indentation (Spaces/Tabs):

```
# Correct - 4 spaces indentation
if True:
    print("This is indented")
    print("This too")

# Wrong - no indentation
if True:
print("This will cause ERROR!")
```

2. Case Sensitivity:

Python Syntax Rules ii

```
name = "John"
Name = "Jane"     # Different variable!
print(name)       # John
print(Name)       # Jane
```

3. No Semicolons Needed:

```
# Python (correct)
print("Hello")
print("World")

# Not needed (but works)
print("Hello");
print("World");
```

Python Syntax Rules iii

4. Comments:

```
# This is a comment
print("Hello")  # Comment at end of line

"""
This is a
multi-line comment
"""
```

Understanding Python Indentation i

Indentation is how Python groups code together

C++ uses braces {}:

```
if (age >= 18) {
    cout << "You can vote!"
    ;
    cout << "You are an
    adult.";
}
cout << "This is outside";
```

Python uses indentation:

```
if age >= 18:
    print("You can vote!")
    print("You are an adult
    .")
print("This is outside")
```

Indentation Rules:

- Use 4 spaces (recommended)
- Be consistent throughout your code
- Python will show `IndentationError` if wrong

Common Mistake

Mixing tabs and spaces causes errors!

Best Practice

Configure your editor to show spaces and use 4 spaces for indentation

Creating Python Files i

Writing your first script:

1. Create a new file: `hello.py`
2. Add this code:

```
# My first Python program
print("Hello, World!")
print("Python is awesome!")
```

3. Save the file
4. Run it: `python hello.py`

Creating Python Files ii

Note

- Lines starting with # are comments
- Comments are ignored by Python
- Use them to explain your code

Variables and Data Types

What are Variables? i

Variables are like labeled boxes that store data

```
name = "Alice"
age = 25
height = 5.6
is_student = True
```

Variable Rules:

- Start with letter or underscore (_)
- Can contain letters, numbers, underscores
- Case-sensitive (`Name` is not `name`)
- No spaces or special characters

What are Variables? ii

Good vs Bad Variable Names

Good: `student_name`, `total_score`, `is_valid`

Bad: `x`, `data`, `thing`, `2names`

Python Data Types i

Type	Description	Example
`int`	Whole numbers	`42, -17, 0`
`float`	Decimal numbers	`3.14, -2.5`
`str`	Text	`"Hello", 'Python'`
`bool`	True/False	`True, False`

```
# Examples of each type
count = 100          # int
price = 19.99        # float
message = "Hello"    # str
is_ready = True      # bool

# Check the type
print(type(count))  # <class 'int'>
```

Working with Integers i

Integers are whole numbers (positive, negative, or zero):

```
# Creating integers
age = 25
year = 2024
negative = -42
zero = 0

# No size limit in Python!
big_num = 123456789012345678901234567890
print(big_num)
```

Key Features:

- No size limits (unlike other languages)
- Can be positive, negative, or zero

Working with Integers ii

- Can handle very large numbers automatically
- Used for counting, indexing, and whole number calculations

Examples

```
students = 30
temperature = -5
population = 8000000000
```

Working with Floats i

Floats are decimal numbers with fractional parts:

```
# Creating floats
price = 19.99
temperature = -3.5
pi = 3.14159

# Scientific notation
large = 1.5e6      # 1,500,000
small = 2.3e-4     # 0.00023
```

Key Features:

- Used for decimal numbers and measurements
- Supports scientific notation (1.5e6)

Working with Floats ii

- Can be positive, negative, or zero
- Used for precise calculations (money, measurements, etc.)

Examples

```
height = 5.9
weight = 68.5
gpa = 3.75
```

Working with Booleans i

Booleans represent True or False values:

```
# Creating booleans
is_student = True
has_license = False
is_raining = False
is_sunny = True
```

Key Features:

- Only two values: True or False (capitalized)
- Used for decision-making and conditions
- Result of comparisons and logical operations
- Essential for controlling program flow

Working with Booleans ii

Examples

```
is_logged_in = True
game_over = False
has_permission = True
```

Working with Strings i

Strings are sequences of characters:

```
# Creating strings
first_name = "John"
last_name = 'Doe'
message = """This is a
multi-line string"""

# Empty string
empty = ""
```

Key Features:

- Use single quotes ('hello') or double quotes ("hello")
- Triple quotes for multi-line strings

Working with Strings ii

- Used for text, names, messages, etc.
- Immutable - cannot be changed after creation

Examples

```
name = "Alice"
city = "New York"
greeting = "Hello, World!"
```

Python Operators

Arithmetic Operators i

Python supports all basic math operations:

```
# Basic arithmetic operators
a = 10
b = 3

print(a + b)     # Addition: 13
print(a - b)     # Subtraction: 7
print(a * b)     # Multiplication: 30
print(a / b)     # Division: 3.333...
print(a // b)    # Integer division: 3
print(a % b)     # Remainder (modulo): 1
print(a ** b)    # Exponentiation: 1000
```

Arithmetic Operators ii

Order of Operations

Python follows PEMDAS (Parentheses, Exponents, Multiplication/Division, Addition/Subtraction)

```
result = 2 + 3 * 4        # 14, not 20
result = (2 + 3) * 4      # 20
```

Comparison Operators i

Used to compare values and return True or False:

Operator	Description	Example
==	Equal to	5 == 5 = True
!=	Not equal to	5 != 3 = True
>	Greater than	7 > 5 = True
<	Less than	3 < 8 = True
>=	Greater than or equal	5 >= 5 = True
<=	Less than or equal	4 <= 6 = True

```
age = 18
print(age >= 18)          # True
print(age == 21)          # False
print("hello" == "hi")    # False
```

Logical Operators i

Used to combine multiple conditions:

Operator	Description
`and`	Both conditions must be True
`or`	At least one condition must be True
`not`	Reverses the Boolean value

```
age = 20
has_license = True

# AND operator
if age >= 18 and has_license:
    print("You can drive!")

# OR operator
```

Logical Operators ii

```
if age < 18 or not has_license:
    print("You cannot drive")

# NOT operator
is_minor = not (age >= 18)
print(is_minor)  # False
```

String Operators i

Special operators for working with strings:

```
# String concatenation with +
first_name = "John"
last_name = "Doe"
full_name = first_name + " " + last_name
print(full_name)  # John Doe

# String repetition with *
greeting = "Hello! "
print(greeting * 3)  # Hello! Hello! Hello!

# Membership operators (in, not in)
sentence = "Python is awesome"
print("Python" in sentence)      # True
print("Java" not in sentence)    # True
```

String Operators ii

Practice

```
password = "secret123"
if len(password) >= 8 and "123" in password:
    print("Password contains numbers")
```

Assignment Operators i

Shortcuts for updating variables:

Basic Assignment:

```
x = 10                    # Assign value
```

Arithmetic Assignment:

```
score = 100
score += 10               # score = score + 10 = 110
score -= 5                # score = score - 5 = 105
score *= 2                # score = score * 2 = 210
score /= 3                # score = score / 3 = 70.0
score //= 2               # score = score // 2 = 35.0
score %= 4                # score = score % 4 = 3.0
score **= 2               # score = score ** 2 = 9.0
```

Assignment Operators ii

String Assignment:

```
name = "John"
name += " Doe"          # name = name + " Doe"
print(name)             # John Doe

greeting = "Hi! "
greeting *= 3           # greeting = greeting * 3
print(greeting)         # Hi! Hi! Hi!
```

Key Features

- Assignment operators provide shortcuts for common operations
- Work with numbers, strings, and other data types
- Make code more concise and readable
- Essential for loops and iterative operations

Assignment Operators iii

Common Usage

```
# Counting
count = 0
count += 1              # Increment by 1

# Building strings
message = "Hello"
message += " World!"     # Add to string
```

Operator Precedence i

Order of operations in Python (high to low priority):

Precedence Order:

1. `()` - Parentheses
2. `**` - Exponentiation
3. `+x`, `-x` - Unary plus/minus
4. `*`, `/`, `//`, `%` - Multiplication, Division
5. `+`, `-` - Addition, Subtraction
6. `<`, `<=`, `>`, `>=`, `==`, `!=` - Comparisons
7. `not` - Logical NOT
8. `and` - Logical AND
9. `or` - Logical OR

Operator Precedence ii

Examples:

```
# Without parentheses
result = 2 + 3 * 4 ** 2
# Order: 4**2=16, 3*16=48, 2+48=50
print(result)          # 50

# With parentheses
result = (2 + 3) * 4 ** 2
# Order: (2+3)=5, 4**2=16, 5*16=80
print(result)          # 80
```

Complex Example:

Operator Precedence iii

```
x = 5
result = x > 3 and x < 10 or x == 0
# Order: x>3=True, x<10=True, True and True=True,
#        x==0=False, True or False=True
print(result)          # True
```

Best Practice

Use parentheses to make your intentions clear: `(x > 3) and (x < 10)`

Type Conversion (Casting)

Converting Between Data Types i

Sometimes you need to convert from one type to another:

To Integer - int():

```
age = int("25")           # "25" -> 25
score = int(95.7)         # 95.7 -> 95 (truncated)
value = int(True)         # True -> 1
value = int(False)        # False -> 0

# This will cause ERROR:
# bad = int("hello")      # ValueError!
```

To Float - float():

```
price = float("19.99")    # "19.99" -> 19.99
num = float(25)           # 25 -> 25.0
pi = float("3.14")        # "3.14" -> 3.14
```

Converting Between Data Types ii

To String - str():

```
score = str(95)              # 95 -> "95"
pi = str(3.14159)            # 3.14159 -> "3.14159"
result = str(True)           # True -> "True"
```

To Boolean - bool():

```
# Numbers: 0 is False, others True
bool(0)          # False
bool(42)         # True
bool(-5)         # True

# Strings: empty is False, others True
bool("")         # False
bool("hello")    # True
bool("0")        # True (string "0", not number)
```

Converting Between Data Types iii

Why Type Conversion Matters i

Common use cases:

```
# Can't mix types in arithmetic
age = "25"
# next_year = age + 1  # ERROR! Can't add string and
    int

# Need to convert first
age = int("25")
next_year = age + 1    # Works! 26

# Can't concatenate string and number
score = 95
# message = "Score: " + score  # ERROR!

# Need to convert first
```

Why Type Conversion Matters ii

```
message = "Score: " + str(score)  # Works! "Score: 95"
```

Remember

Python won't automatically convert between incompatible types. You must explicitly convert when needed!

Getting User Input

Interactive Programs with input() i

Making programs interactive:

```
# Getting user input
name = input("What's your name? ")
print(f"Hello, {name}!")

# Input is always a string
age_str = input("How old are you? ")
age = int(age_str)  # Convert to integer
print(f"Next year you'll be {age + 1}")
```

Important!
The `input()` function always returns a string, even if the user types numbers. Use `int()`, `float()`, etc. to convert.

Basic Input/Output

Formatting Output i

Different ways to format output:

```
name = "Alice"
age = 25
score = 95.5

# Method 1: Concatenation
print("Name: " + name + ", Age: " + str(age))

# Method 2: Multiple arguments
print("Name:", name, "Age:", age)

# Method 3: f-strings (recommended)
print(f"Name: {name}, Age: {age}")
print(f"Score: {score:.1f}%")  # One decimal place
```

Formatting Output ii

```
# Method 4: format() method
print("Name: {}, Age: {}".format(name, age))
```

f-string Formatting

- :.2f - 2 decimal places
- :10 - minimum width of 10
- :>10 - right-aligned in width 10

Handling Different Input Types i

Robust input handling:

```
# Getting different types of input
name = input("Enter your name: ")
age = int(input("Enter your age: "))
height = float(input("Enter your height in meters: "))

# Handling yes/no input
response = input("Do you like Python? (y/n): ")
if response == 'y':
    print("Great choice!")
else:
    print("Give it a try!")
```

Conditional Statements

Making Decisions with if Statements i

Programs need to make decisions:

```
# Basic if statement
age = 18
if age >= 18:
    print("You can vote!")

# if-else statement
temperature = 25
if temperature > 30:
    print("It's hot outside!")
else:
    print("Nice weather!")
```

Comparison Operators:

Making Decisions with if Statements ii

- == (equal), != (not equal)
- >, <, >=, <=
- `in`, `not in` (for membership)

Multiple Conditions with elif i

Checking multiple conditions:

```
score = 85

if score >= 90:
    grade = "A"
elif score >= 80:
    grade = "B"
elif score >= 70:
    grade = "C"
elif score >= 60:
    grade = "D"
else:
    grade = "F"

print(f"Your grade is: {grade}")
```

Multiple Conditions with elif ii

Real-world Example

```
weather = input("How's the weather? (sunny/rainy/cold)
    : ")

if weather == "sunny":
    print("Wear sunglasses!")
elif weather == "rainy":
    print("Take an umbrella!")
elif weather == "cold":
    print("Wear a jacket!")
else:
    print("Enjoy your day!")
```

Logical Operators i

Combining conditions:

```
# AND operator - both conditions must be true
age = 20
has_license = True

if age >= 18 and has_license:
    print("You can drive!")

# OR operator - at least one condition must be true
day = "Saturday"
is_holiday = False

if day == "Saturday" or day == "Sunday" or is_holiday:
    print("No work today!")
```

Logical Operators ii

```
# NOT operator - reverses the condition
is_raining = False
if not is_raining:
    print("Let's go for a walk!")
```

Introduction to Loops

Why Do We Need Loops? i

Problem: Print numbers 1 to 5

Without loops (repetitive):

```
print(1)
print(2)
print(3)
print(4)
print(5)
```

With loops (efficient):

```
for i in range(1, 6):
    print(i)
```

Why Do We Need Loops? ii

Key Point

Loops help us repeat code efficiently without writing the same thing multiple times!

Types of Loops in Python

Python has two main types of loops:

for Loop

- Used when you know how many times to repeat
- Iterates over sequences
- Most common in Python

while Loop

- Used when condition-based repetition
- Continues until condition is False
- Need to be careful with infinite loops

Today's Focus
We'll master both types and learn when to use each one!

For Loops

The for Loop i

Basic syntax:

```
for variable in sequence:
    # code to repeat
    # this code is indented
```

Using range():

```
# Print numbers 0 to 4
for i in range(5):
    print(i)

# Output:
# 0
# 1
# 2
```

The for Loop ii

```
# 3
# 4
```

How range() Works

- `range(5)` generates: 0, 1, 2, 3, 4
- `range(start, stop)` - from start to stop-1
- `range(start, stop, step)` - with custom step

range() Function Examples i

Different ways to use range():

```
# range(stop) - starts at 0
for i in range(5):
    print(i)            # 0, 1, 2, 3, 4

# range(start, stop)
for i in range(2, 6):
    print(i)            # 2, 3, 4, 5

# range(start, stop, step)
for i in range(0, 10, 2):
    print(i)            # 0, 2, 4, 6, 8

# Counting backwards
for i in range(10, 0, -1):
```

range() Function Examples ii

```
    print(i)              # 10, 9, 8, ..., 1
```

Iterating Over Sequences i

for loops work with any sequence:

Iterating over strings:

```
name = "Python"
for letter in name:
    print(letter)
# Output: P y t h o n (each on new line)
```

Iterating over lists:

Iterating Over Sequences ii

```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(f"I like {fruit}")
# Output:
# I like apple
# I like banana
# I like cherry
```

Practice

```
# Calculate sum of numbers 1 to 10
total = 0
for num in range(1, 11):
    total += num
print(f"Sum: {total}")  # Sum: 55
```

Nested for Loops i

Loops inside loops:

```
# Multiplication table
for i in range(1, 4):
    for j in range(1, 4):
        print(f"{i} x {j} = {i*j}")
    print()  # Empty line after each table

# Output:
# 1 x 1 = 1
# 1 x 2 = 2
# 1 x 3 = 3
#
# 2 x 1 = 2
# 2 x 2 = 4
# 2 x 3 = 6
```

Nested for Loops ii

```
#  ...
```

Nested Loops: Pattern Printing i

Creating patterns with nested loops:

```
# Print a triangle of stars
for i in range(1, 6):
    for j in range(i):
        print("*", end="")
    print()  # New line

# Output:
# *
# **
# ***
# ****
# *****
```

Nested Loops: Pattern Printing ii

Challenge
Try creating a square pattern or a reverse triangle!

While Loops

The while Loop i

Syntax:

```
while condition:
    # code to repeat
    # runs as long as condition is True
```

Basic example:

```
count = 1
while count <= 5:
    print(count)
    count += 1  # IMPORTANT: update the condition!

# Output: 1 2 3 4 5
```

The while Loop ii

Warning!
Always make sure the condition eventually becomes False, or you'll have an infinite loop!

Common use case - user input validation:

```
password = ""
while password != "secret":
    password = input("Enter password: ")
print("Access granted!")
```

for vs while: When to Use Which? i

Use for loop when:

- You know the number of iterations
- Iterating over a sequence (list, string, range)
- More common in Python

```
# Good use of for loop
for i in range(10):
    print(i)
```

Use while loop when:

- Condition-based repetition
- Unknown number of iterations

for vs while: When to Use Which? ii

- Waiting for specific event/input

```
# Good use of while loop
guess = 0
while guess != 7:
    guess = int(input("Guess the number: "))
```

Loop Control Statements

break Statement i

Exit the loop immediately:

```
# Find the first number divisible by 7
for num in range(1, 100):
    if num % 7 == 0:
        print(f"First number divisible by 7: {num}")
        break  # Exit loop
# Output: First number divisible by 7: 7
```

With while loop:

break Statement ii

```
while True:  # Infinite loop
    response = input("Continue? (y/n): ")
    if response == 'n':
        break  # Exit loop
    print("Continuing...")
print("Loop ended")
```

Use Case
Use break when you want to exit a loop before it naturally completes.

continue Statement i

Skip current iteration, continue with next:

```
# Print only odd numbers
for num in range(1, 11):
    if num % 2 == 0:
        continue  # Skip even numbers
    print(num)
# Output: 1 3 5 7 9
```

Practical example:

continue Statement ii

```
# Process only valid inputs
numbers = [1, -5, 3, -2, 7, 0, 4]
positive_sum = 0

for num in numbers:
    if num <= 0:
        continue  # Skip non-positive numbers
    positive_sum += num
print(f"Sum of positive numbers: {positive_sum}")
# Output: Sum of positive numbers: 15
```

else Clause with Loops i

else executes when loop completes normally (not broken):

```
# Check if a number is prime
num = 17
for i in range(2, num):
    if num % i == 0:
        print(f"{num} is not prime")
        break
else:
    # This runs if loop wasn't broken
    print(f"{num} is prime")
```

With while loop:

else Clause with Loops ii

```
count = 0
while count < 5:
    print(count)
    count += 1
else:
    print("Loop completed normally")
```

Putting It All Together

Complete Example: Simple Calculator i

```
# Simple Calculator Program
print("Simple Calculator")
print("Choose operation: +, -, *, /")

operation = input("Enter operation: ")
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))

if operation == '+':
    result = num1 + num2
elif operation == '-':
    result = num1 - num2
elif operation == '*':
    result = num1 * num2
elif operation == '/':
    if num2 != 0:
```

Complete Example: Simple Calculator ii

```
        result = num1 / num2
    else:
        result = "Error: Cannot divide by zero!"
else:
    result = "Invalid operation"

print(f"Result: {result}")
```

Practice Exercise: Personal Info Program

Your turn! Create a program that:

1. Asks for user's name, age, and favorite color
2. Calculates birth year
3. Creates a personalized message
4. Determines if they can vote (age >= 18)

Personal Info Program - Solution i

```
# Get user input
name = input("What's your name? ")
age = int(input("How old are you? "))
color = input("What's your favorite color? ")

# Calculate birth year
current_year = 2024
birth_year = current_year - age

# Create messages
print(f"Hello {name}! You were born in {birth_year}.")
print(f"I see you like {color}. That's a great color!"
    )

# Check voting eligibility
if age >= 18:
```

Personal Info Program - Solution ii

```
    print("You are eligible to vote!")
else:
    years_to_vote = 18 - age
    print(f"You can vote in {years_to_vote} years.")
```

Week 1 Summary

What We Learned i

- **Programming Concepts:** What programming is and how computers work
- **Python Basics:** Why Python, installation, and running programs
- **Variables & Data Types:** Storing and working with different types of data
- **Python Operators:** Arithmetic, comparison, logical, and string operators
- **User Input:** Making programs interactive with `input()`
- **Conditionals:** Making decisions with `if/elif/else`
- **Input/Output:** Formatting and displaying information
- **Loops:** Repeating code with `for` and `while` loops
- **Loop Control:** Using `break`, `continue`, and `else`

What We Learned ii

Key Programming Concepts

- Breaking problems into small steps
- Writing clear, readable code
- Testing your programs
- Debugging when things go wrong

Practice Assignments i

Complete these exercises before next class:

1. **Temperature Converter:** Convert Celsius to Fahrenheit
2. **Grade Calculator:** Calculate average of 3 test scores
3. **Simple Quiz:** Ask 3 questions and give a score
4. **Age Categories:** Classify age into child/teen/adult/senior
5. **BMI Calculator:** Calculate and categorize BMI

Programming Tips

- Start small and build up
- Test frequently
- Use meaningful variable names
- Add comments to explain your code
- Don't be afraid to experiment!

Next Week Preview

Week 2: Data Structures

- **Lists:** Ordered, mutable collections
- **Strings:** Text processing and manipulation
- **Tuples:** Immutable sequences
- **Dictionaries:** Key-value pairs
- **Sets:** Unique element collections
- **Choosing the Right Data Structure**

Questions?
Feel free to ask any questions about today's topics!

Thank You!

Thank you for your attention!

Keep Practicing!

Access Course Materials:

Download Course Materials

Center for Artificial Intelligence & Emerging Technologies