Notes: Mastering Input and Output (I/O) in Python

Welcome to your comprehensive guide on handling input and output (I/O) in Python. In programming, I/O is the bridge between your code and the user (or other systems). It's how your program "talks" and "listens."

- **Output**: This is how your program presents information. It could be showing a message to the user, displaying a calculation result, or saving data to a file.
- **Input**: This is how your program receives information. It's typically data that the user types in from their keyboard.

We will explore Python's built-in tools that make these operations simple yet powerful.

Part 1: The print() Function - Your Program's Voice

The most common way for a Python program to "speak" is by using the print() function. It displays whatever you put inside its parentheses on the screen.

Basic Printing

You can print simple text (strings), numbers, and other data types.

```
# example_print_basic.py

# Printing a string of text. Strings must be enclosed in single ('') or double ("") quotes.
print("Hello, World! This is my first Python output.")

# Printing a whole number (integer)
print(123)

# Printing a number with a decimal point (float)
print(3.14159)

# You can also print the result of a calculation
print("The sum of 5 and 3 is:", 5 + 3)
```

Printing Multiple Items at Once

You can pass multiple items to the print() function, separating them with commas. By default, print() will add a single space between each item.

```
# example_print_multiple.py

name = "Alex"
age = 25
city = "New York"

# The print function will automatically add spaces between the items.
print("Name:", name, "Age:", age, "City:", city)

# Output will be:
# Name: Alex Age: 25 City: New York
```

Customizing print() with sep and end

The print() function has special arguments that give you more control over the output. The two most important are sep (separator) and end.

- **sep**: Controls what character is used to separate the items you are printing. The default is a space (' ').
- **end**: Controls what character is printed at the very end. The default is a newline character ('\n'), which is why every print() statement usually starts on a new line.

```
# example_print_sep_end.py

# --- Using the 'sep' argument ---
# Let's separate items with a hyphen instead of a space.
print("apple", "banana", "cherry", sep="-")

# Let's use a comma and a space.
print("one", "two", "three", sep=", ")


# --- Using the 'end' argument ---
# Let's print two statements on the same line.
# The first print statement will end with a space instead of a newline.
print("This is the first part", end=" ")
print("and this is the second part.")

# You can use any string for 'end'.
print("Loading", end="...")
print("Complete!")

# Combining both sep and end
print("file", "user", "documents", sep="/", end=".txt\n")

# Output of the above code:
# apple-banana-cherry
# one, two, three
# This is the first part and this is the second part.
# Loading...Complete!
# file/user/documents.txt
```

Formatting Your Output

Just printing variables can be messy. It's often better to format your strings for a cleaner, more readable output. Python offers several ways to do this.

1. f-Strings (The Modern & Recommended Way)

Formatted string literals, or "f-strings," are the easiest and most powerful way to embed expressions inside string literals. You just prefix the string with an f or F and write variable names or expressions inside curly braces {}.

```
# example_fstrings.py

item = "Laptop"
price = 1200.50
quantity = 3

# Using an f-string to create a clean output string.
# The variables inside the curly braces are automatically replaced with their values.
receipt = f"Item: {item}, Quantity: {quantity}, Unit Price: ${price}"
print(receipt)

# You can even do calculations or call functions inside the braces!
total_cost = f"Total Cost: ${price * quantity}"
print(total_cost)

# You can also format numbers.
# Here, ':.2f' means format the number as a float with exactly two decimal places.
pi = 3.14159265
print(f"The value of Pi, formatted to 2 decimal places, is {pi:.2f}")

# Output of the above code:
# Item: Laptop, Quantity: 3, Unit Price: $1200.5
# Total Cost: $3601.5
# The value of Pi, formatted to 2 decimal places, is 3.14
```

2. The str.format() Method (A More Traditional Way)

Before f-strings, the format() method was the standard. It works by placing placeholder curly braces {} in the string and then calling the .format() method on the string with the variables you want to insert.

```
# example_str_format.py

name = "Jane"
language = "Python"

# The values passed to .format() will fill the placeholders in order.
greeting = "Hello, my name is {} and I am learning {}.".format(name, language)
print(greeting)

# You can also use numbers inside the braces to specify the order.
reordered_greeting = "I am learning {1} and my name is {0}.".format(name, language)
print(reordered_greeting)

# You can also use keywords to make it even clearer.
keyword_greeting = "My name is {n} and my language is {l}.".format(n="Jane", l="Python")
print(keyword_greeting)

# Output of the above code:
# Hello, my name is Jane and I am learning Python.
# I am learning Python and my name is Jane.
# My name is Jane and my language is Python.
```

Part 2: The input() Function - Your Program's Ears

Now that your program can talk, let's teach it how to listen. The input() function pauses your program and waits for the user to type something and press the Enter key.

Basic Input

You can provide a "prompt" inside the input() function's parentheses. This is a message that tells the user what you want them to enter.

CRUCIAL POINT: The input() function **always** returns the user's entry as a **string**, even if they type in numbers.

```
# example_input_basic.py

# The string inside input() is the prompt shown to the user.
print("What is your name?")
user_name = input() # The program will wait here for the user to type.

# A more common way is to put the prompt directly inside the input function.
user_city = input("What city do you live in? ")

# Now we can use the variables that stored the user's input.
print(f"Hello, {user_name}! It's nice to know you live in {user_city}.")

# Example interaction:
# What is your name?
# David
# What city do you live in? London
# Hello, David! It's nice to know you live in London.
```

Converting Input: Type Casting

Because input() always gives you a string, you can't do math with it directly. You must first convert the string to the correct numeric type. This conversion is called **Type Casting**.

- int(): Converts a string to an integer (a whole number).
- float(): Converts a string to a float (a number with a decimal).

```
# example_type_casting.py

# Get the user's age. The value in 'age_str' will be a string.
age_str = input("How old are you? ")

# Get the user's height. The value in 'height_str' will be a string.
height_str = input("What is your height in meters? ")

# Now, we convert them.
# The int() function will cause an error if the user types "twenty" instead of 20.
```

```
user_age = int(age_str)

# The float() function handles decimals.
user_height = float(height_str)

# Now we can perform calculations with the converted values.
age_in_a_decade = user_age + 10

print(f"You are currently {user_age} years old.")
print(f"In a decade, you will be {age_in_a_decade} years old.")
print(f"You are {user_height} meters tall.")

# You can also combine the input and casting in one line for efficiency.
user_weight = float(input("What is your weight in kg? "))
print(f"Your weight is {user_weight} kg.")
```

Handling Multiple Inputs on One Line

Sometimes you want the user to enter multiple values on a single line (e.g., "10 20 30"). You can achieve this by using the .split() string method, which splits a string into a list of smaller strings.

```
# example_multiple_inputs.py

# Ask the user for three numbers, separated by spaces.
# The prompt makes it clear how the input should be formatted.
data = input("Enter three numbers separated by spaces: ") # User might type "5 10 15"

# The .split() method breaks the string '5 10 15' into a list of strings: ['5', '10', '15']
pieces = data.split()

# Now we can access each piece by its index and convert it to an integer.
num1 = int(pieces[0])
num2 = int(pieces[1])
num3 = int(pieces[2])

# Perform a calculation with the numbers.
total_sum = num1 + num2 + num3

print(f"The numbers you entered were: {num1}, {num2}, and {num3}.")
print(f"Their sum is: {total_sum}.")

# Example interaction:
# Enter three numbers separated by spaces: 25 50 100
# The numbers you entered were: 25, 50, and 100.
# Their sum is: 175.
```

By mastering print() and input(), you have learned the two most fundamental operations for creating interactive programs in Python.