Notebook 1.1 - Variables & Basic Data Types

Management Science

Introduction

Welcome to your first interactive Python tutorial!

The Coffee Shop Calculator Problem

Imagine you just started working at a busy coffee shop called “Bean Counter.” On your first day, the manager asks you to help with various calculations: pricing drinks, applying discounts, calculating tips, and formatting receipts. The old calculator is broken, and they need a better system anyway. Lucky for them, you know Python!

In this tutorial, we’ll learn about variables and basic data types to build your own coffee shop calculator system. By the end, you’ll be able to handle any calculation the shop throws at you!

> **i** Note
>
> If a cell is marked with YOUR CODE BELOW, you are expected to write your code in that cell.

Section 1 - Variables as Named Storage

Variables are like labeled containers that store information. In our coffee shop, we need to keep track of prices, customer names, order quantities, and more. Variables allow us to store this information in a way that we can easily access and update it later.

```
# Variables are created by giving them a name and a value
coffee_price = 4.50
customer_name = "Alex"
cups_ordered = 3
shop_open = True

print(f"Customer {customer_name} ordered {cups_ordered} coffees at
${coffee_price} each")
```

```
Customer Alex ordered 3 coffees at $4.5 each
```

Why Use Variables?

- Reusability: Once you store a value in a variable, you can use it multiple times without rewriting the value.
- Readability: Variables make your code easier to read and understand.
- Flexibility: You can easily update the value of a variable without changing the entire code.

! Variable Naming Rules

- Use descriptive names (coffee_price is better than cp)
- Use lowercase with underscores for multiple words
- Can't start with numbers
- Can't use Python keywords (like if, for, def)

Exercise 1.1 - Your First Variables

Create three variables for a new order:

- `latte_price` should be 5.25
- `customer` should be "Maria"
- `quantity` should be 2
- `shop_closed` = False

```
# YOUR CODE BELOW
```

```
# Test your answer
assert latte_price == 5.25, "The latte_price should be 5.25"
assert customer == "Maria", "The customer should be 'Maria'"
assert quantity == 2, "The quantity should be 2"
assert shop_closed == False, "The shop should be open"
print("Perfect! You've created your first variables for the coffee shop!")
```

Exercise 1.2 - Updating Variables

Variables can change! A customer changed their mind and wants 4 lattes instead of 2. Update the `quantity` variable to 4 and create a new variable `total_order` that stores the total price (latte_price multiplied by quantity).

```
# YOUR CODE BELOW
# Assume latte_price = 5.25 and quantity = 2 from previous exercise
# Tip: Just overwrite these old values
# Then calculate: total_order = latte_price * quantity
```

```
# Test your answer
assert quantity == 4, "The quantity should be updated to 4"
assert total_order == 21.0, "The total_order should be 21.0 (5.25 * 4)"
print("Excellent! You can update variables and create new ones from
existing values!")
```

Section 2 - Data Types: Numbers and Text

Python has different types of data. The three most common are:

- int (integers): Whole numbers like 1, 42, –10
- float (floating-point): Decimal numbers like 3.14, 2.50, –0.5
- str (strings): Text like “coffee”, “Bean Counter”, “Order #123”
- bool (boolean): True or False

```
# Different data types in our coffee shop
cups_sold = 150           # int - whole number
price = 4.99              # float - decimal number
shop_name = "Bean Counter" # str - text
is_open = True            # bool - true or false

# Python can tell you the type of any variable
print(f"cups_sold is a {type(cups_sold)}")
print(f"price is a {type(price)}")
print(f"shop_name is a {type(shop_name)}")
print(f"is_open is a {type(is_open)}")
```

```
cups_sold is a <class 'int'>
price is a <class 'float'>
shop_name is a <class 'str'>
is_open is a <class 'bool'>
```

Why Different Data Types?

- Precision: Different data types allow for different levels of precision. For example, floats can represent decimal numbers, while integers cannot.
- Operations: Different data types support different operations. For example, you can add numbers but concatenate strings.
- Memory: Different data types use different amounts of memory. For example, integers use less memory than floats.

⚠ Warning

Common Mistake: Mixing up numbers and text that looks like numbers!

```
quantity = "5"  # This is text, not a number!
quantity * 2    # This would give "55" not 10!
```

Exercise 2.1 - Identifying Data Types

Create variables of each type for our coffee shop:

- `daily_customers` (int): 85 customers visited today
- `average_tip` (float): 2.50 dollars average tip
- `best_seller` (str): "Caramel Macchiato"

```
# YOUR CODE BELOW
```

```
# Test your answer
assert daily_customers == 85 and type(daily_customers) == int,
"daily_customers should be the integer 85"
assert average_tip == 2.50 and type(average_tip) == float, "average_tip
should be the float 2.50"
assert best_seller == "Caramel Macchiato" and type(best_seller) == str,
"best_seller should be the string 'Caramel Macchiato'"
print("Great job! You understand the different data types!")
```

Exercise 2.2 - Type Conversion

Sometimes we need to convert between types. A customer's loyalty card bonus points were entered as text "1234" but we need it as a number to do calculations.

Convert the string `card_points = "1234"` to an integer and create a new variable `new_points` that adds 100 bonus points to it.

💡 Tip

You can use the `int()` function to convert a string to an integer or use the `float()` function to convert a string to a float.

```
# YOUR CODE BELOW
card_points = "1234"
```

```
# Test your answer
assert card_points == 1234, "card_points should be 1234 as an integer"
assert new_points == 1334, "new_points should be 1334 (1234 + 100)"
print("Perfect! You can convert between data types!")
```

Section 3 - Basic Arithmetic Operations

Now let's use Python's arithmetic operations to handle coffee shop calculations!

```
# Basic arithmetic operations
espresso_price = 3.50
shots = 2

# Addition and multiplication
double_espresso = espresso_price * shots
with_tip = double_espresso + 1.50

# Division and subtraction
change = 20.00 - with_tip
price_per_shot = double_espresso / shots

print(f"Double espresso: ${double_espresso}")
print(f"With tip: ${with_tip}")
print(f"Change from $20: ${change}")
print(f"Price per shot: ${price_per_shot}")
```

```
Double espresso: $7.0
With tip: $8.5
Change from $20: $11.5
Price per shot: $3.5
```

> Common Errors in Arithmetic
>
> - TypeError: This occurs when you try to perform an operation on incompatible types, like adding a string to a number.
> - ZeroDivisionError: This occurs when you try to divide by zero.

> ! Python Arithmetic Operators
>
> - `+` Addition
> - `-` Subtraction
> - `*` Multiplication
> - `/` Division
> - `**` Exponentiation (power)
> - `//` Floor division (rounds down)
> - `%` Modulo (remainder)

Exercise 3.1 - Calculate the Bill

A group orders:

- 3 cappuccinos at $4.75 each
- 2 muffins at $3.50 each

- 1 sandwich at $8.95

Calculate the `subtotal`, then add 19% tax to get the `total_with_tax`.

> Tip
>
> To add 19% tax to a value, you can multiply by 1.19.

```
# YOUR CODE BELOW
```

```
# Test your answer
import math
assert math.isclose(subtotal, 30.20, rel_tol=0.01), "Subtotal should be
30.20"
assert math.isclose(total_with_tax, 30.20*1.19, rel_tol=0.01), "Total with
tax should be 35.94"
print("Excellent calculation! You've mastered basic arithmetic!")
```

Exercise 3.2 - Splitting the Bill

The group wants to split the bill evenly among 4 people. Calculate:

- How much each person pays (`per_person`)
- If they pay with exact change, how many dollars (`dollars`) and cents (`cents`) each person needs. Provide each as variable.

> Tip
>
> You could use `//` for floor division and `%` for remainder, but you don't need to do that as other solutions are possible. If you want to round the cents, you can use the `round()` function.

```
# YOUR CODE BELOW
# Assume total_with_tax = 32.616 from previous exercise
```

```
# Test your answer
assert math.isclose(per_person, 8.154, rel_tol=0.01), "Each person should
pay 8.154"
assert dollars == 8, "Each person needs 8 dollars"
assert cents == 15, "Each person needs 15 cents (rounded)"
print("Great work! You can split bills and handle money calculations!")
```

Section 4 - F-strings for Output Formatting

F-strings make it easy to create formatted output - perfect for receipts and displays!

```python
# F-strings let you embed variables and expressions in text
item = "Flat White"
price = 4.25
quantity = 2
total = price * quantity

# Basic f-string
receipt = f"Item: {item}, Quantity: {quantity}, Total: ${total}"
print(receipt)

# Formatting numbers
formatted_receipt = f"""
╔═══════════════════════════╗
║     BEAN COUNTER          ║
╠═══════════════════════════╣
║ {item:<16} {quantity:2d}      ║
║ Price: ${price:>6.2f}           ║
║ Total: ${total:>6.2f}           ║
╚═══════════════════════════╝
"""
print(formatted_receipt)
```

```
Item: Flat White, Quantity: 2, Total: $8.5

╔═══════════════════════════╗
║     BEAN COUNTER          ║
╠═══════════════════════════╣
║ Flat White        2       ║
║ Price: $  4.25            ║
║ Total: $  8.50            ║
╚═══════════════════════════╝
```

Common Errors in F-strings

- SyntaxError: This occurs when you forget to use an `f` before the string or use incorrect syntax within the curly braces.
- TypeError: This occurs when you try to format a value that is not compatible with the specified format.

! F-string Formatting:

- `{variable:.2f}` - Show 2 decimal places
- `{variable:10}` - Use 10 characters of space
- `{variable:<10}` - Left align in 10 spaces
- `{variable:>10}` - Right align in 10 spaces
- `{variable:^10}` - Center in 10 spaces

Exercise 4.1 - Create a Simple Receipt

Create a formatted receipt for a coffee order with these variables:

- `coffee_type = "Americano"`
- `size = "Large"`
- `price = 3.75`

Create a variable called `receipt` with the format: "Order: [size] [coffee_type] - $[price with 2 decimal places]"

> Tip
>
> Use an f-string with `{price:.2f}` to format the price with exactly 2 decimal places.

```
# YOUR CODE BELOW
```

```
# Test your answer
assert receipt == "Order: Large Americano - $3.75", "Receipt should be
'Order: Large Americano - $3.75'"
print("Perfect! You've created your first formatted string!")
```

Conclusion

Congratulations! You've successfully learned the fundamentals of Python programming!

You've learned:

- Variables - Storing and updating information
- Data Types - Working with integers, floats, and strings
- Arithmetic - Performing calculations
- F-strings - Creating formatted output

You can now handle any calculation at Bean Counter coffee shop! From calculating bills and splitting checks to creating professional receipts, you have all the basic tools you need.

Remember:

- Variables are named containers for storing data
- Python has different data types (int, float, str) for different kinds of information
- Arithmetic operations follow standard mathematical rules
- F-strings make it easy to create formatted, professional output

Next up: In the next tutorial, we'll learn about lists and loops to handle multiple orders at once and analyze daily sales data!

Bibliography