

Python Refresher

INFO 153B/253B: Backend Web Architecture

Kay Ashaolu - Instructor

Aishwarya Sriram - TA

What Is in This Lesson?

- Overview of Python Variables
- String Formatting Techniques
- User Input and Conversion
- Core Data Structures (Lists, Tuples, Sets, Dictionaries)
- Conditional Statements and Loops
- Function Basics, Parameters, and Return Values

Variables in Python

- A variable is a “name” for a value
- **Example:**



```
1 x = 15  
2 price = 9.99
```

- Variable references a location in memory
- Right side (value) is created first, then left side (name) assigned

Variables in Depth

- Variables are references, not strictly boxes
- Types:
 - `int` (e.g., 1500)
 - `float` (e.g., 9.99)
 - `str` (e.g., "Hello")
- Reassignment:



```
1 name = "Rolf"  
2 name = "Bob" # Now `name` points to "Bob"
```

Variables in Action



```
1 discount = 0.2
2 price = 9.99
3 result = price * (1 - discount)
4 print(result) # 7.992
```

- Shows variable usage, arithmetic, and `print` function
- `price * (1 - discount)` is evaluated, then stored

String Formatting: f-Strings

- Available in Python 3.6+
- Embed variables directly:

```
1 name = "Bob"  
2 greeting = f"Hello, {name}"  
3 print(greeting) # Outputs: Hello, Bob
```

- Simplifies building user-friendly strings

String Formatting:

`str.format()`

- Template approach:

```
1 greeting = "Hello, {}!"  
2 with_name = greeting.format("Rolf")  
3 print(with_name) # Hello, Rolf!
```

- Useful when reusing templates with multiple placeholders:

```
1 message = "Hello, {}. Today is {}."  
2 print(message.format("Alice", "Monday"))
```

Getting User Input

- `input()` reads a string from the console
- Example:



```
1 name = input("Enter your name: ")
2 print(f"Nice to meet you, {name}!")
```

- Always returns a `str`, even if the user types numbers

Combining Input & Type Conversion

- Convert string to int/float:



```
1 user_input = input("Enter a number: ")
2 number = int(user_input)
3 print(number * 2)
```

- Example usage:



```
1 square_feet = int(input("Size in sq ft: "))
2 sq_metres = square_feet / 10.8
3 print(f"{square_feet} sq ft is {sq_metres:.2f} sq m")
```

Writing Our First Python App

- Simple age-to-months example:



```
1 user_age = int(input("Enter your age: "))
2 months = user_age * 12
3 print(f"Your age, {user_age}, equals {months} months.")
```

Lists, Tuples, and Sets

- **Lists:** Ordered, mutable, [1, 2, 3]
- **Tuples:** Ordered, immutable, (1, 2, 3)
- **Sets:** Unordered, no duplicates, { "a", "b" }
- Choose based on your data's needs

Lists vs. Tuples vs. Sets

- **Lists:**

- Append/remove elements freely



```
1 friends = ["Bob", "Rolf"]  
2 friends.append("Anne")
```

- **Tuples:**

- Fixed once created



```
1 names = ("Bob", "Anne")  
2 # Can't modify!
```

- **Sets:**

- Fast membership tests ("Bob" in my_set)
- No duplicate items

Manipulating Lists in Python

- Subscript notation: `friends[0]` for the first element
- Adding/removing:



```
1 friends = ["Bob", "Rolf"]
2 friends.append("Anne")
3 friends.remove("Bob")
```

- Lists are heavily used for ordered collections

Example: List & Indices



```
1 grades = [35, 67, 98, 100, 100]
2 total = sum(grades)
3 average = total / len(grades)
4 print(f"Average grade: {average}")
```

- Summation and length demonstrate list utility

Booleans & Comparisons

- True / False
- Common operators: ==, !=, >, <, >=, <=
- The `is` keyword checks if two references are the exact same object:

```
1 a = [1, 2]
2 b = [1, 2]
3 print(a == b) # True
4 print(a is b) # False
```

If Statements



```
1 day = input("Enter the day: ").lower()
2
3 if day == "monday":
4     print("Have a great start to your week!")
5 elif day == "tuesday":
6     print("Keep going!")
7 else:
8     print("Full speed ahead!")
```

- Indentation defines blocks
- `if/elif/else` chain covers multiple conditions

Nested If & Elif

- More complex checks:



```
1 if age >= 18:
2     if has_permission:
3         print("Access granted")
4     elif age >= 21:
5         print("Special privileges for 21+!")
6     else:
7         print("Permission denied")
8 else:
9     print("Underage")
```

Loops: While

- Repeat code while condition is `True`



```
1 user_input = ""
2 while user_input != "n":
3     user_input = input("Play again? (Y/n): ")
4     if user_input == "n":
5         print("Goodbye!")
```

- **Caution:** Must ensure condition eventually becomes false or break out

Loops: For

- Iterate over collections:



```
1 # Iterating over a list
2 friends = ["Bob", "Rolf", "Anne"]
3 for friend in friends:
4     print(f"{friend} is my friend")
5
6 # Using range() with default start=0
7 for i in range(3):
8     print(i)
9 # Outputs: 0, 1, 2
```

- Use `range()` to loop a certain number of times

List Comprehensions

- Concise way to transform lists



```
1 numbers = [1, 2, 3]
2 doubled = [x * 2 for x in numbers]
3 # doubled -> [2, 4, 6]
```

- Syntax: [`<expression>` for `<var>` in `<iterable>`]

List Comprehensions with Conditionals



```
1 friends = ["Sam", "Samantha", "Bob", "Anne"]
2 starts_s = [friend for friend in friends if friend.startswith("S")]
3 print(starts_s) # ["Sam", "Samantha"]
```

- Filter elements using an `if` clause

Dictionaries

- Key-value mapping:



```
1 friend_ages = {  
2     "Rolf": 24,  
3     "Adam": 30,  
4     "Anne": 27  
5 }
```

- Access via keys: `friend_ages["Adam"]` -> 30

Iterating Over Dictionaries



```
1 for name, age in friend_ages.items():  
2     print(f"{name} is {age} years old.")
```

- `.items()` returns key-value pairs
- `.values()` returns just values
- `.keys()` returns just keys

Functions in Python



```
1 def say_hello():  
2     print("Hello!")
```

- Define using `def` keyword
- Call with `say_hello()`
- Allows code reuse & organization

Function Parameters & Return Values



```
1 def add(x, y):  
2     return x + y  
3  
4 result = add(5, 8)  
5 print(result) # 13
```

- **Positional arguments:** `add(5, 8)`
- **Keyword arguments:** `add(x=5, y=8)`
- If no `return`, function returns `None` by default

Summary

- **Python Refresher:** Variables, data structures, conditionals, loops, functions
- **Why It Matters:**
 1. Forms the foundation for backend logic
 2. Essential for handling data & building RESTful APIs

Questions?