Class Two Theory

1. Recap of Old Concepts

1. **Interpreter vs. Compiler**
    - An **interpreter** (like Python's) executes code line by line at runtime.
    - A **compiler** (like C/C++ compilers) converts source code to machine code entirely before execution.
2. **Dynamic vs. Static Typing**
    - **Dynamic typing** (Python): Variable types are determined at runtime and can change over the program's execution.
    - **Static typing** (C/C++/Rust): Variable types are checked and fixed at compile time.
3. **High-level vs. Low-level Languages**
    - **High-level languages** (Python, Java): More abstraction from machine code, easier syntax, but generally slower.
    - **Low-level languages** (Assembly, C): Closer to hardware, less abstraction, typically faster.

2. Casting (Type Conversion)

Casting is the process of converting one data type to another in Python. This is often done when dealing with user input or when you need to perform numeric operations on values that are strings by default.

- **int()**: Converts a value to an integer (e.g., `int("2")` -> `2`).
- **float()**: Converts a value to a float (e.g., `float("2.2")` -> `2.2`).
- **str()**: Converts a value to a string (e.g., `str(2)` -> `"2"`).

Example:

```python
a = int(2.9)    # a = 2
b = float("3.2")  # b = 3.2
```

3. Using `input()`

- The built-in `input()` function **always returns a string**.
- If you want a numeric value, you must cast it to `int` or `float`.

Example:

```
user_input = input("Enter a number: ")   # Always a string
num = int(user_input)                    # Convert to integer
print(num + 10)
```

4. Multiple Assignment

In Python, you can assign multiple variables in one line:

```
x, y = 2, 4
```

- `x` will get `2`
- `y` will get `4`

You can also do parallel assignment:

```
a, b = b, a
```

(This swaps the values of `a` and `b` without using a temporary variable.)

5. Basic Data Types

4. **int**: For integers (e.g., `5` , `-10`).
5. **float**: For floating-point numbers (e.g., `3.14` , `2.0`).
6. **str**: For strings (e.g., `"Hello"` , `"2.2"`).
7. **bool**: For booleans (`True` or `False`).

Remember, Python's **dynamic typing** allows you to change a variable's type by simply assigning it a new value of a different type.

6. Generating Random Numbers

- Python provides the **`random`** module for generating pseudo-random numbers.
- Common function:

  ```
  import random
  x = random.randrange(0, 10)  # Generates a random integer from 0 to 9
  print(x)
  ```

7. String Manipulation

7.1 Slicing

- You can slice strings using the `[start:end]` syntax:

```
text = "Hello"
print(text[0:2])  # "He"
print(text[:2])   # "He" (same as above, start = 0 by default)
print(text[1:])   # "ello" (ends at the last character by default)
```

7.2 Changing Case

- `text.lower()` -> converts all characters to lowercase.
- `text.upper()` -> converts all characters to uppercase.
- `text.capitalize()` -> capitalizes the first character, rest are lowercased.

7.3 Stripping Whitespace

- `text.strip()` -> removes leading and trailing whitespace.
- Example:

```
string = "   Hello World   "
print(string.strip())  # "Hello World"
```

7.4 Replacing and Splitting

- `text.replace("old", "new")` -> replaces all occurrences of `"old"` with `"new"` .
- `text.split("separator")` -> splits a string into a list of substrings. If no `separator` is provided, it defaults to splitting on whitespace.

7.5 Concatenation

- You can join strings using the `+` operator.

```
a = "Hello"
b = "World"
c = a + " " + b  # "Hello World"
```

8. Checking Object Size

- The `sys` module provides `sys.getsizeof(object)`, which returns the size of an object in bytes.
- Usage:

```python
import sys
text = "Hello"
print(sys.getsizeof(text))
```

This can be helpful in understanding memory usage for strings or other objects.