Class one code notes

> *Yaar i am sharing the code in the form of notes, so you could rewrite and practice the code on your own , rather then just copying that code into your own file,*

1. Compiler vs Interpreter

- **Compiler (C, C++, Rust, etc.)**
  A compiler converts **all** source code into machine code (binary) before executing it. This typically results in faster execution since the translation happens once and then the program runs directly on the machine.
- **Interpreter (Python, JavaScript, etc.)**
  An interpreter reads the source code **line by line** and executes it on the fly. This generally results in slower execution compared to compiled languages, because the translation happens continuously during runtime.
- **Which one is faster: C or Python?**
  In most cases, C is faster because it is compiled to machine code directly and has a lower level of abstraction. Python, being interpreted and dynamically typed, usually has overhead that makes it slower.
- **Static vs Dynamic Typing**
  - *C/C++/Rust:* Statically typed (types are checked at compile time).
  - *Python:* Dynamically typed (types are checked at runtime).

---

2. Basic Examples in Python

Variables and Print Statements

```python
# Example of character variables
x = 'A'
y = 'B'

# Print statements
print("Hello World")
print(x)
print(y)  # Printing the variable y
```

Control Flow (Indentation)

```
if x == y:
    print("x is equal to y")
    print("Continue")
    print("Continue")
    if x == y:
        print("Continue again")
```

---

3. Statement Endings in Python

- Unlike languages like C, C++, or Java, **Python does not require a semicolon** at the end of each statement.
- However, you **can** use semicolons in Python to separate statements on the same line. For example:

```
a = 5; b = 11; c = 10
```

Though valid, it’s more readable to place each statement on its own line.

Line Continuation

- A backslash (`\`) can be used to continue a line of code onto the next line:

```
sum_value = 1 + 2 + 3 + 4 + 5 \
            + 6 + 7 + 8 + 9 + 10
print(sum_value)
```

- Alternatively, parentheses can be used:

```
total = (
    1 + 2 + 3 + 4 +
    5 + 6 + 7 + 8 +
    9 + 10
)
print(total)
```

---

4. Operator Precedence

- Python follows standard mathematical precedence:
    1. `()`
    2. `*, /, //, %`

3. `+, -`

Example:

```
z = 2 + 3 + (1 + 2)
print(z)  # 2 + 3 + 3 = 8

f = 2 - 2 + 3 * 2
print(f)  # 0 + 6 = 6
```

---

5. Comments in Python

- **Single-line comment:** Use `#`
- **Multi-line comment:** Use triple quotes `""" ... """`

```
# This is a single-line comment.

"""
This is a multi-line comment.
It can span multiple lines.
"""
```

---

6. Identifiers and Naming Conventions

- Python is **case-sensitive**:

  ```
  x = 100
  x = 120
  X = 130  # Different from lowercase x
  print(x)  # 120
  print(X)  # 130
  ```

- **Valid variable names** can include letters, digits, and underscores, but **cannot** start with a digit.

  ```
  Ab = 10
  ab = 20
  aB = 30
  # All of these are different variables
  ```

- **Keywords** (like `if` , `for` , `True` , `False` , etc.) cannot be used as variable names.

Common Naming Styles

1. **snake_case:** `car_one`
2. **camelCase:** `carOne`
3. **ALL_CAPS:** `CAR_ONE`

---

7. Expressions, Operators, and Operands

- An **expression** is a piece of code that returns a value. Example:

  ```
  x = 2 + 3  # 2 + 3 evaluates to 5
  ```

- **Arithmetic Operators:** `+` , `-` , `*` , `/` , `//` (floor division), `%` (modulus), `**` (exponent)
- **Comparison Operators:** `==` , `>` , `<` , `>=` , `<=` , `!=`
- **Logical Operators:** `and` , `or` , `not`

---

8. Taking Input from the User

```
name = input("What is your name? ")
print("The name is:", name)
```

- `input()` always returns a string in Python. You can convert it to another type (e.g., `int` , `float`) as needed.

Complete Example

```
# Demonstration of the concepts:

x = 'A'
y = 'B'

print("Hello World")
print("x:", x)
print("y:", y)

if x == y:
    print("x is equal to y")
    print("Continue")
    if x == y:
        print("Continue again")
```

```python
# Multiple statements in one line (not recommended in practice):
a = 5; b = 11; c = 10

# Using backslash for line continuation:
sum_value = 1 + 2 + 3 + 4 + 5 \
            + 6 + 7 + 8 + 9 + 10
print("Sum with backslash:", sum_value)

# Using parentheses for line continuation:
total = (
    1 + 2 + 3 + 4 +
    5 + 6 + 7 + 8 +
    9 + 10
)
print("Sum with parentheses:", total)

# Operator precedence example:
z = 2 + 3 + (1 + 2)
f = 2 - 2 + 3 * 2

print("z:", z)
print("f:", f)

"""
Multi-line comment:
Variables, naming, expressions, etc.
"""

# Variable reassignment and case sensitivity:
x = 100
x = 120
X = 130
print("x:", x)  # 120
print("X:", X)  # 130

# Input example:
name = input("What is your name? ")
print("The name is:", name)
```