Day 3

1.0 Introduction: Understanding Keywords in Python

Keywords are reserved words in Python that have a special meaning in the language. Traditionally, they cannot be used as variable or function names. Python has 35 of these **hard keywords**.

Python also introduced **soft keywords**. Soft keywords behave like regular words unless used in a specific context. This allows Python to add new syntax without breaking existing code that might have used those words as names.

2.0 Hard vs. Soft Keywords

Hard Keywords

Hard keywords are always reserved. You cannot use them as identifiers. For example:

```
# ❌ This will raise a SyntaxError
if = 5
```

Soft Keywords

Soft keywords only act as keywords in certain contexts. Outside of those contexts, you can use them as variable or function names. Python currently has four soft keywords: `_`, `case`, `match`, and `type`.

```
# ✅ This is valid
match = 5
print(match)  # Output: 5
```

You can see all keywords programmatically using Python's `keyword` module:

```
import keyword

print(keyword.kwlist)      # List of hard keywords
```

```
print(keyword.softkwlist)  # List of soft keywords
```

Hard Keywords	Soft Keywords
import keyword	`_`, `case`, `match`, `type`

3.0 Soft Keywords in `match-case` Statements

The `match-case` statement (Python 3.10+) uses `match`, `case`, and `_`:

- `match`: Starts the pattern matching block.
- `case`: Defines a pattern to match against.
- `_`: Acts as a wildcard for unmatched patterns.

Example:

```
def get_action(scenario: int):
    if scenario == 1:
        return "Hello"
    elif scenario == 2:
        return 10
    elif scenario == 3:
        return 3 + 4j
    else:
        return [1, 2, 3]

action = get_action(4)

match action:
    case "Hello":
        print("Hello Learner")
    case 10:
        print("My lucky number is 10")
    case 3 + 4j:
        print("This is a complex number")
    case _:
```

```
        print("Default trap: matched a non-specific pattern.")
```

Here, `match`, `case`, and `_` only act as keywords inside the `match` block. Outside, you can use them freely as identifiers.

4.0 Soft Keyword `type` for Type Aliases

Python introduced `type` as a soft keyword for defining **type aliases**, which improves readability in type hints.

Syntax:

```
type AliasName = ComplexType
```

Example:

```
# Simple type alias
type Point = tuple[float, float]
p1: Point = (3.0, 4.5)
print(type(p1))  # Output: <class 'tuple'>

# Composite type alias
type ListOfPoints = list[Point]
my_list: ListOfPoints = [(3.0, 4.5), (5.0, 9.0), (1.0, -0.5)]
print(my_list)
```

Using `type` makes complex type hints easier to read and maintain.