**Python Notes**

Python is a widely popular, high-level, general-purpose programming language. It's known for several key characteristics:

* **Readability:** Python's syntax emphasizes code readability with its use of whitespace and English-like keywords. This makes it beginner-friendly and easy to learn.
* **Versatility:** Python is used in a vast range of fields, including:
  + Web development
  + Data science and machine learning
  + Scripting and automation
  + Education
  + System administration
* **Extensive Libraries:** Python has a massive collection of standard libraries and third-party modules offering pre-written code for various tasks. This saves you time and effort.
* **Strong Community:** Python boasts a supportive and active community, providing excellent documentation, tutorials, and help forums.

**Why is it Called "Python"?**

The name "Python" has a rather amusing origin. Guido van Rossum, the creator of Python, was a fan of the British comedy group **Monty Python's Flying Circus**. While developing the language, he wanted a name that was short, unique, and slightly mysterious, and settled on "Python".

**Keywords:**

* **Reserved Words**: Keywords are predefined words in a programming language that have special meaning for the compiler or interpreter. They are the building blocks of a language's syntax.
* **Cannot Be Used as Variable Names**: You cannot use keywords as names for variables, functions, or other elements you define in your code.

**Examples of Python Keywords:**

* **Data types**: int, float, bool, str
* **Control flow**: if, else, elif, for, while, break, continue
* **Defining Structures**: class, def, return
* **Others:** import, as, from, try, except, lambda, global

**Identifiers:**

* **User-Defined Names:** Identifiers are names you create to represent different elements in your code, such as:
  + Variables (e.g., student\_name, total\_score)
  + Functions (e.g., calculate\_average, display\_data)
  + Classes (e.g., Employee, Product)
  + Modules (e.g., my\_functions.py)

**Rules for Creating Identifiers**

* **Case-Sensitive**: Python is case-sensitive (total is different from Total).
* **Start with Letter or Underscore**: The first character must be a letter (A-Z, a-z) or an underscore (\_).
* **Remaining Characters**: Subsequent characters can be letters, numbers, or underscores.
* **No Keywords Allowed**: You can't use a reserved keyword as an identifier.

**Python comments?**

Comments are portions of code that the Python interpreter intentionally ignores during execution.

They are designed for human readers, not the computer.

**Why use comments?**

* **Explain your code:** Comments help you and others understand the logic and reasoning behind your coding choices.
* **Improve readability:** Well-written comments make your code much easier to maintain, even months or years after you initially wrote it.
* **Debugging:** You can temporarily "comment out" sections of code to isolate problems.
* **Collaboration:** Comments help you work with other developers effectively.

**Types of comments in Python**

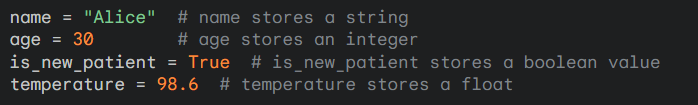
1. **Single-line comments:**
   * Begin with a hash symbol (#)
   * Everything after the # on that line is ignored
   * Example: # This is a single-line comment
2. **Multiline comments:**
   * While Python doesn't have a dedicated multiline comment syntax, there are two common ways to achieve it:
     + **Multiple single-line comments:** Precede each line with #.
     + **Triple-quoted strings:** Enclose the comment text within triple single quotes (''') or triple double quotes ("""). If the string isn't assigned to a variable, the interpreter ignores it.

**What are variables?**

* Variables are named containers that store data values in your program. Think of them as boxes with labels where you can put different items.
* In Python, you don't need to explicitly declare the variable's data type before assigning a value. Python determines the data type automatically based on the value you assign.

**How to create variables in Python**

1. **Choose a descriptive name:** Follow these rules:
   * Must start with a letter (a-z, A-Z) or an underscore (\_).
   * Can contain letters, numbers, and underscores.
   * Case-sensitive (age and Age are different).
2. **Use the assignment operator (**=**):** To assign a value to a variable.

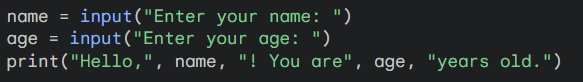


**Key Points**

* **Dynamic typing:** A variable in Python can hold different data types throughout the program. For example, you can reassign name to store a number later.
* **Overwriting values:** When you assign a new value to an existing variable, the old value is replaced.

**Python Input :**

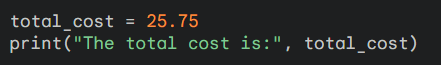
* **Gathering data from the user:** Python allows you to interact with the user and collect information during program execution.
* **The**input()**function:**
  + Displays a prompt (optional) to the user on the console.
  + Waits for the user to type something and press Enter.
  + Captures whatever the user typed and returns it as a string.



**Important Note:** The input() function always returns a string, even if the user enters a number. You might need to convert the input to an integer or float using int() or float().

**Python Output**

* **Displaying information to the user:** Python provides ways to send the results of your program or other messages to the user.
* **The**print()**function:**
  + Displays information on the console.
  + Can handle strings, numbers, variables, and expressions.
  + Automatically adds a newline character at the end (unless you change this behavior).



**What are operators?**

Operators are special symbols that perform operations on values (operands) within your code. Python provides a range of operators that fall into the following categories:

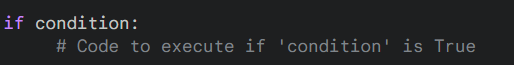
* **Arithmetic Operators**
  + Used for mathematical calculations:
    - Addition (+)
    - Subtraction (-)
    - Multiplication (\*)
    - Division (/)
    - Floor Division (//) - Returns the integer quotient
    - Modulo (%) - Returns the remainder
    - Exponentiation (\*\*) - Raises number to a power
* **Comparison Operators**
  + Used to compare values and return a Boolean (True or False) result:
    - Equal to (==)
    - Not equal to (!=)
    - Greater than (>)
    - Less than (<)
    - Greater than or equal to (>=)
    - Less than or equal to (<=)
* **Logical Operators**
  + Combine Boolean expressions:
    - And (and)
    - Or (or)
    - Not (not)
* **Assignment Operators**
  + Assign values to variables:
    - Basic assignment (=)
    - Compound assignments:
      * Add and assign (+=)
      * Subtract and assign (-=)
      * And so on...
* **Bitwise Operators**
  + Work on the bit-level representation of numbers:
    - Bitwise AND (&)
    - Bitwise OR (|)
    - Bitwise XOR (^)
    - Bitwise NOT (~)
    - Left shift (<<)
    - Right Shift (>>)
* **Membership Operators**
  + Test if a value is contained within a sequence (like a string, list, or tuple):
    - in
    - not in
* **Identity Operators**
  + Compare if two objects occupy the same memory location:
    - is
    - is not

**What are Control Statements?**

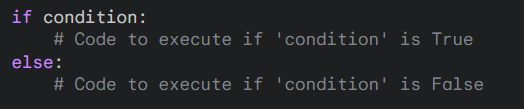
* Control statements are instructions that allow you to alter the normal sequential flow of execution in your program.
* They help you make decisions, execute code blocks conditionally, and create loops.

**Decision-Making: if, if-else, if-elif-else:**

These statements let your Python code make choices based on conditions.

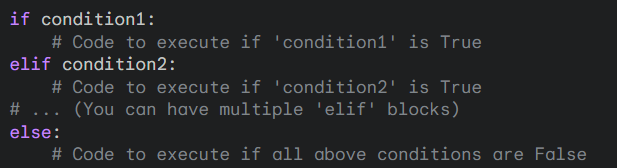
**if statement:**

* **Syntax:**
* **How It Works:** The code inside the if block will only execute if the condition evaluates to True.

**if-else statement**

* **Syntax:**
* **How It Works:** If the condition is True, the first code block executes. If the condition is False, the code in the else block executes.

**if-elif-else statement**

* **Syntax:**
* **How It Works:** Python checks each condition in order. If a condition evaluates to True, its corresponding code block is executed, and the rest of the chain is skipped. If none of the conditions are True, the code in the else block is executed.

**For Loops**

* **Iterating over sequences:** for loops are primarily used to iterate over a sequence of items (like strings, lists, tuples, etc.).
* **Syntax:**
* **How it works:**
  + For each item in the sequence, the variable item takes on its value.
  + The code block inside the loop is executed once for each value.

**While Loops**

* **Repeating based on a condition:** while loops continue to execute a block of code as long as a certain condition remains True.
* **Syntax:**
* **How it works:**
  + The condition is checked.
  + If the condition is True, the code block executes.
  + The process repeats (back to step 1) until the condition becomes False.

**Key Differences**

* **Knowing iterations beforehand:** Use a for loop when you know how many times you need to iterate in advance.
* **Condition-based repetition:** Use a while loop when you want the loop to continue as long as a certain condition holds true, and you're not sure how many iterations will be necessary beforehand.

**Important Note:** Be careful with while loops. If the condition never becomes False, you'll end up with an infinite loop!

**Python's *break* and *continue* statement*:***

***break***

* **Purpose:** Used to immediately terminate a loop, regardless of whether the loop condition is still True.
* **Effect:** Execution jumps to the code immediately following the loop.

***continue***

* **Purpose:** Used to skip the current iteration of a loop and move to the beginning of the next iteration.
* **Effect:** The remaining code in the current iteration is skipped, and the loop continues if the condition is still True.

**Truthy and Falsy Values:**

In Python, all objects have an inherent Boolean value, either True or False. This concept is called "truthiness" or "falsiness."

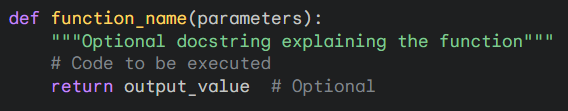
* **Truthy Values:** Most objects are considered Truthy (interpreted as True in conditional statements).
  + Examples:
    - Non-empty strings ("hello")
    - Non-zero numbers (1, -5.2)
    - Non-empty lists, tuples, dictionaries
    - Instances of custom classes (in most cases)
* **Falsy Values:** A specific set of objects are considered Falsy (interpreted as False in conditional statements). These include:
  + None
  + False
  + Zero of any numeric type (0, 0.0, 0j)
  + Empty sequences: "", [], ()
  + Empty mappings: {}

**Why Does This Matter?**

Python uses truthy and falsy values extensively in control flow statements:

**Python function:**

* A function is a block of reusable code designed to perform a specific task or calculation.
* Functions help modularize your programs, making your code more organized and easier to maintain.

**Syntax 🡪**

**Key Components**

* def**:** The keyword that tells Python you're defining a function.
* function\_name**:** A descriptive name you give to your function.
* parameters (Optional): Values the function can receive as input.
* docstring (Optional): A string enclosed in triple quotes to document what the function does.
* return (Optional): Sends a value back to the code that called the function.

**Why use functions?**

* **Code Reusability:** Write code once and use it multiple times, reducing duplication.
* **Modularity:** Break down a complex program into smaller, manageable units.
* **Testability:** Test functions independently.
* **Improved Readability:** Functions with meaningful names make your code more self-explanatory.

**Types of functions:**

**1. Built-in Functions**

* These functions are pre-defined in the Python language and ready to use.
* **Examples:**
  + print() - outputs information to the console.
  + len() - calculates the length of a sequence or collection.
  + type() - returns the data type of an object.
  + int(), float() - convert values to integers or floats.

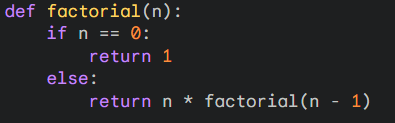
**2. User-Defined Functions**

* These are the functions you create! They allow you to encapsulate specific tasks and calculations, promoting code reusability
* **Defined using the**def**keyword.**

**3. Lambda Functions**

* Small, anonymous functions created on the fly using the lambda keyword.
* Often used for short, simple operations where a full function definition would be unnecessary.
* **Example:**

**4. Recursive Functions**

* Functions that call themselves within their definition.
* Commonly used to solve problems that can be broken down into smaller, self-similar subproblems.
* **Example (Calculating factorial):**

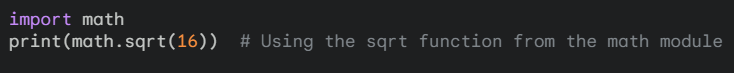
**Import keyword:**

The import keyword is essential for using code from other files or modules in your Python programs. It gives you access to a library of pre-written functionality, making development easier and more efficient.

**How it Works**

1. **Modules:** Python code is often organized into modules, which are files with the .py extension. A module can contain classes, functions, and variables.
2. **The**import**Statement:** You use import to bring code from a module into your current script.

**Importing an entire module:**



**Importing specific items:**



**Renaming with 'as'**:



* **Standard Library:** Python comes with an extensive standard library of modules that provide functionality for file I/O, networking, data processing, and much more.
* **Third-party Libraries:** Thousands of external libraries are available through PyPI (the Python Package Index), offering features from web development to scientific computing.
* **Organization:** Imports help keep your project organized by separating code into logical units.