Python Reference Guide

Module 1 – Python Fundamentals

This document provides a **clear, structured, and detailed explanation** of the core Python concepts covered in **Module 1**. These concepts form the foundation for writing, reading, and understanding Python code used in cybersecurity tasks.

---

1. Comments

What are Comments?

Comments are **notes written by programmers** to explain what the code does. Python **ignores comments**, meaning they do not affect program execution.

Syntax

This is a comment

Example

Print approved usernames

print("bmoreno")

Why Comments Matter

- Improve code readability

- Explain the purpose of code
- Helpful for teams and future reference

---

2. Functions

Functions are **built-in operations** that perform specific tasks. Python provides many built-in functions that are commonly used in cybersecurity scripts.

`print()` Function

Purpose: Displays output on the screen

print("login success")

Output:

login success

print(9 < 7)

Output:

False

`type()` Function

Purpose: Returns the data type of a value

print(type(51.1))

Output:

<class 'float'>

print(type(True))

Output:

<class 'bool'>

`range()` Function

Purpose: Generates a sequence of numbers

range(0, 5, 1)

- Start: 0 (inclusive)
- Stop: 5 (exclusive)
- Step: 1

Generated sequence:

0, 1, 2, 3, 4

range(5)

- Default start: 0
- Default step: 1

Generated sequence:

0, 1, 2, 3, 4

---

3. Conditional Statements

Conditional statements allow Python to **make decisions** based on conditions.

`if` Statement

Used to evaluate a condition.

```
if device_id != "la858zn":
    print("Access denied")


if user in approved_list:
    print("Access granted")
```

`elif` Statement

Evaluated **only if previous conditions are False**.

```
elif status == 500:
    print("Server error")
```

`else` Statement

Executed when **all previous conditions are False**.

```
else:
    print("Unknown status")
```

Logical Operators

`and`

Both conditions must be True

```
if username == "bmoreno" and login_attempts < 5:
    print("Login allowed")
```

`or`

Only one condition must be True

```
if status == 100 or status == 102:
    print("Processing")
```

`not`

Negates a condition

```
if not account_status == "removed":
    print("Account active")
```

---

4. Iterative Statements

Iterative statements are used to **repeat actions**, which is essential for handling logs, users, and security events.

`for` Loop

Used to iterate over a sequence.

```
for username in ["bmoreno", "tshah", "elarson"]:
    print(username)
```

```
for i in range(10):
    print(i)
```

`while` Loop

Repeats code **as long as a condition remains True**.

```
while login_attempts < 5:
    print("Attempting login")
    login_attempts += 1
```

Loop Control Statements

`break`

Immediately exits the loop

```
if attempts == 3:
    break
```

`continue`

Skips the current iteration

```
if user == "guest":
    continue
```

Summary

- **Comments** explain code
- **Functions** perform actions
- **Conditional statements** control decision-making
- **Iterative statements** repeat tasks efficiently

These concepts are essential for **automation, monitoring, and analysis** in real-world programming scenarios.

Python Fundamentals: Module 1 Reference Guide

This document offers a **clear, structured, and detailed reference** for the foundational Python concepts covered in **Module 1**. Mastering these elements is crucial for reading, writing, and understanding Python code, particularly in the context of cybersecurity.1. Code Annotations (Comments)

Comments are **programmer-written notes** used to explain code logic. Python **disregards comments** during execution, meaning they have no impact on the program's operation.

Aspect	Description	Syntax/Example
Purpose	Improve readability, explain code's intent, aid collaboration and future maintenance.	
Syntax	Starts with the # symbol.	`# This is a comment`
Example	`print("bmoreno") # Print approved username`	

2. Standard Operations (Functions)

Functions are **pre-defined operations** designed to carry out specific tasks. Python includes numerous built-in functions frequently used in cybersecurity scripts.

Function	Purpose	Example	Output
`print()`	Displays output to the console.	`print("login success")`	`login success`
		`print(9 < 7)`	`False`
`type()`	Returns the data type of a given value.	`print(type(51.1))`	`<class 'float'>`
		`print(type(True))`	`<class 'bool'>`
`range()`	Generates a sequence of numbers.	`range(0, 5, 1)`	Sequence: 0, 1, 2, 3, 4 (Start: 0 incl., Stop: 5 excl., Step: 1)
		`range(5)`	Sequence: 0, 1, 2, 3, 4 (Default Start: 0, Default Step: 1)

3. Decision Making (Conditional Statements)

Conditional statements enable Python programs to **make decisions** by evaluating specific conditions.

Statement	Purpose	Context	Example
`if`	Evaluates a primary condition.	Always evaluated first.	`if device_id != "la858zn": print("Access denied")`

elif	Evaluated **only if** preceding `if`/`elif` conditions are False.	Used for secondary checks.	`elif status == 500: print("Server error")`
else	Executed when **all** preceding `if`/`elif` conditions are False.	Provides a fallback action.	`else: print("Unknown status")`

Logical Operators

Operator	Meaning	Condition	Example
and	Both conditions must be True.	Strict check.	`if username == "bmoreno" and login_attempts < 5: print("Login allowed")`
or	At least one condition must be True.	Lenient check.	`if status == 100 or status == 102: print("Processing")`
not	Negates the result of a condition (True becomes False, False becomes True).	Inversion.	`if not account_status == "removed": print("Account active")`

4. Repetitive Actions (Iterative Statements)

Iterative statements are vital for **repeating tasks** and are essential for handling common cybersecurity operations like analyzing logs, processing lists of users, or monitoring security events.

Loop Type	Purpose	Execution Logic	Example
`for` Loop	Iterates over items in a sequence (e.g., list, range).	Executes once for each item in the sequence.	`for username in ["bmoreno", "tshah"]: print(username)`
			`for i in range(10): print(i)`
`while` Loop	Repeats a block of code **as long as** a specified condition remains True.	Continues until the condition becomes False.	`while login_attempts < 5: print("Attemptin g login"); login_attempts += 1`

Loop Control Statements

Statement	Action	Effect	Example
`break`	Stops the loop immediately.	Exits the entire loop structure.	`if attempts == 3: break`
`continue`	Skips the rest of the code in the current iteration.	Proceeds to the next iteration of the loop.	`if user == "guest": continue`

Summary of Module 1 Fundamentals

Concept	Primary Role	Application
Comments	Code explanation and documentation.	Improving code clarity.
Functions	Executing predefined tasks.	Performing standard operations.

Conditional Statements	Controlling program flow based on conditions.	Decision-making.
Iterative Statements	Repeating actions efficiently.	Automation, monitoring, and data analysis.

These core concepts are indispensable for successful **automation, monitoring, and analysis** in real-world programming and security scenarios.