🐍 Python Mastery Journey - Deep Dive Notes

🚀 Stage 1 - Fundamentals with Challenges, Pro Tips & Hidden Gems

---

✅ Task 1: Print with `sep` and `end`

```python
Student = "Aman"
Age = 21
Score = 92

print("Student", Student, end=" || ", sep=":")
print("Age", Age, end=" || ", sep=":")
print("Score", Score, end=" || ", sep=":")
```

💎 Hidden Python Gems:

- `sep` changes separator between arguments in `print()`
- `end` changes what gets printed at the end (default is `\n`)

🧠 Pro Tips:

- Combine emojis and text for fun CLI outputs
- Use `.center()` and `.ljust()` for UI-like formatting

✅ Task 2: Input Tricks & Emoji Bio Card

```python
width = 80
print("🫷 Bio Card Generator 🫸".center(width))

name = input("😎 Enter your name 👉 ")
age = int(input("👨‍👩‍👦 Enter your age 👉 "))
language = input("🤷 Enter your Language 👉 ")
goal = input("🎯 Enter your goal 👉 ")
experience = input("🧩 Enter your Experience 👉 ")

print("😎 Name".ljust(10), "⇛", name)
print("👨‍👩‍👦 Age".ljust(10), "⇛", age)
print("🤷 Language".ljust(10), "⇛", language)
print("🎯 Goal".ljust(10), "⇛", goal)
print("🧩 Experience".ljust(10), "⇛", experience)
```

💎 Hidden Python Gems:

- `.ljust(width)` aligns text left — perfect for labels

- Emojis + format = CLI swag
- `input().split()` to take multiple values in one line
- `list(map(int, input().split()))` to convert them to a list of integers
- Use `input() or "default"` to apply default values
- Use `from getpass import getpass` to hide password inputs
- Use `inputimeout` from `inputimeout` module to set a timeout for inputs

```
# Example: Multi input in 1 line
x, y, z = map(int, input("Enter 3 numbers: ").split())

# Example: List from input
marks = list(map(int, input("Enter marks: ").split()))

# Example: Default value fallback
name = input("Enter name: ") or "Guest"

# Example: Password input (hidden)
from getpass import getpass
pwd = getpass("Enter your password: ")

# Example: Timeout input
from inputimeout import inputimeout, TimeoutOccurred
try:
    user_input = inputimeout(prompt='You have 5 seconds: ', timeout=5)
except TimeoutOccurred:
    user_input = 'Timed Out'
```

🧠 Pro Tips:

- Input prompts with emojis = more fun
- Use `.center()` and `.ljust()` for elegant terminal UI
- Use `getpass()` when dealing with passwords securely
- Always validate multi-input formats

☑ Task 3: Smart Score Checker (Ternary + Try)

```
try:
    name = input("Enter your name → ")
    p, c, b = map(int, input("Enter marks [P, C, B] → ").split())
    total = p + c + b
    avg = total / 3

    print(f"📊 Total: {total}/300")
    print(f"📈 Average: {avg:.2f}")

    passed = "✅ PASS" if avg >= 40 and all(m >= 33 for m in [p, c, b]) else "❌
FAIL"
    print(passed)
    grade = "🅰️ A" if avg >= 90 else "🅱️ B" if avg >= 75 else "🆎 C" if avg >=
60 else "No Grade"
    print("🎖️ Grade:", grade)

except ValueError:
    print("⚠️ Please enter valid numbers!")
```

💎 Hidden Python Gems:

- `try-except` for safe input (no crash)
- `all()` for checking multiple pass conditions
- Ternary inside print = clean one-liner logic
- `f"{value:.2f}"` formats float to 2 decimal places

🧠 Pro Tips:

- Add input validation in every real-world project
- Use ternary for readability but not for complex logic
- `try-except` + `while` loop = best for retryable inputs
- Format floats in output using `f"{value:.2f}"` to keep them clean and professional

---

✅ Task 4: Loop Mastery - From Basics to Pro

```
# For loop basics
for i in range(5):
    print("🚀", i)

# Range variations
for i in range(1, 10, 2):
    print("⏩", i)

# While loop basics
count = 0
while count < 5:
    print("💧", count)
    count += 1

# Looping over collections
fruits = ['apple', 'banana', 'cherry']
for fruit in fruits:
    print(f"🍏 {fruit}")

# Loop with else
for i in range(3):
    print(i)
else:
    print("✅ Loop finished!")
```

```python
# Hidden Gems
for i in reversed(range(3)):
    print("⬅️", i)

for idx, val in enumerate(['a', 'b', 'c']):
    print(idx, "⇒", val)

for a, b in zip([1, 2, 3], ['a', 'b', 'c']):
    print(a, b)

# Loop Bio Generator Challenge
import time
name = input("Name ➡️ ")
age = input("Age ➡️ ")
hobbies = input("Enter 3 hobbies ➡️ ").split(',')

print(f"\n🧠 Bio of {name}, Age: {age}")
for h in hobbies:
    print("🎯 Hobby ➡️", h.strip())
    time.sleep(1)
```

💎 Hidden Python Gems:

- `range(start, end, step)` for advanced control
- `enumerate()` = cleaner indexing
- `zip()` to combine multiple lists
- `reversed(range(...))` = loop backwards
- `else` block in loops (runs only if not broken)

🧠 Pro Tips:

- Prefer `for item in list` over `for i in range(len(list))`
- Avoid infinite `while` loops unless needed
- Use `break` to exit early, `continue` to skip
- Loop with `else` is a hidden gem for post-loop status

Sure! Below is the **MD text** formatted for your document, covering the topics **One-Line Loops (List Comprehension)**, **Inline Try-Except**, and **all() and any() for Logic Checks**:

🧑‍💻 code block & indentation

🔹 One-Line Loops (List Comprehension)

List comprehensions are a powerful way to create lists in just one line of code. They provide a compact and readable alternative to using loops for list creation.

Syntax:

```
new_list = [expression for item in iterable if condition]
```

Example:

Creating a list of squares of numbers from 1 to 5:

```
squares = [i*i for i in range(1, 6)]
print(squares)  # Output: [1, 4, 9, 16, 25]
```

You can also use an `if` condition:

```
evens = [i for i in range(10) if i % 2 == 0]
print(evens)  # Output: [0, 2, 4, 6, 8]
```

◇ 🧪 Inline Try-Except

Sometimes you may want to handle potential errors inline without needing full error-handling blocks. Python allows a more compact approach with `try-except` inside a single line.

Example:

```
try:
    age = int(input("Enter your age: "))
except:
    age = 0  # Default value in case of an error
```

This way, you can quickly handle any potential input errors (like entering a non-numeric value) and provide a fallback or default value in case of failure.

◇ all() and any() for Logic Checks

Both `all()` and `any()` are useful built-in functions to simplify logical checks across collections.

`all()`:

Returns `True` if **all** elements of the iterable are `True`, otherwise returns `False`.

Example:

Check if all subject marks are above 33:

```
marks = [p, c, b]
if all(m >= 33 for m in marks):
    print("✅ Pass")
else:
    print("❌ Fail")
```

`any()`:

Returns `True` if **any** element of the iterable is `True`, otherwise returns `False`.

Example:

Check if **any** subject marks are below 33:

```
marks = [p, c, b]
if any(m < 33 for m in marks):
    print("❌ Fail")
else:
    print("✅ Pass")
```
