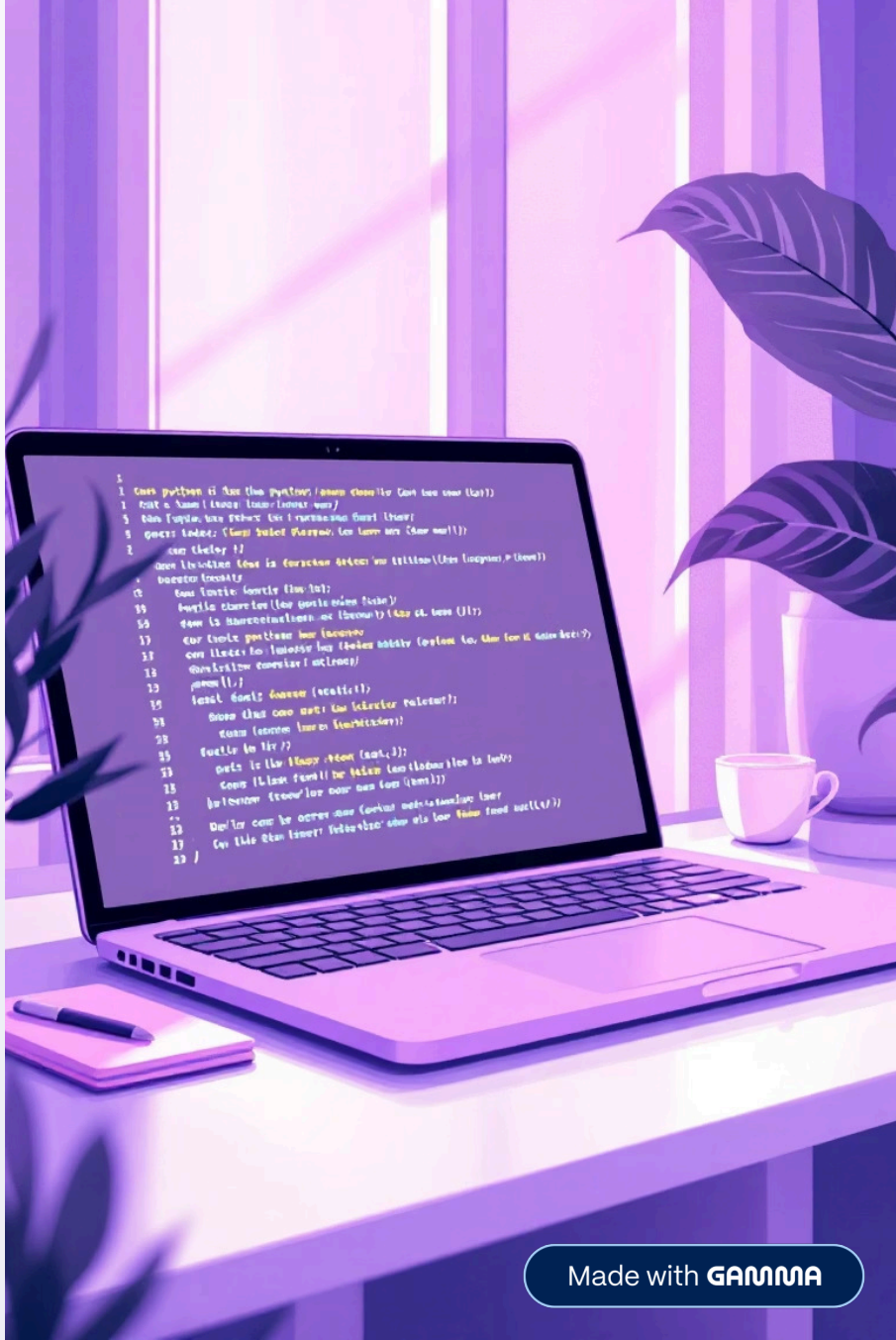
Python Programming Module 1

Abhishek Patil

Department of Computer Science and Design

VTU First Year Engineering — Comprehensive Study Notes

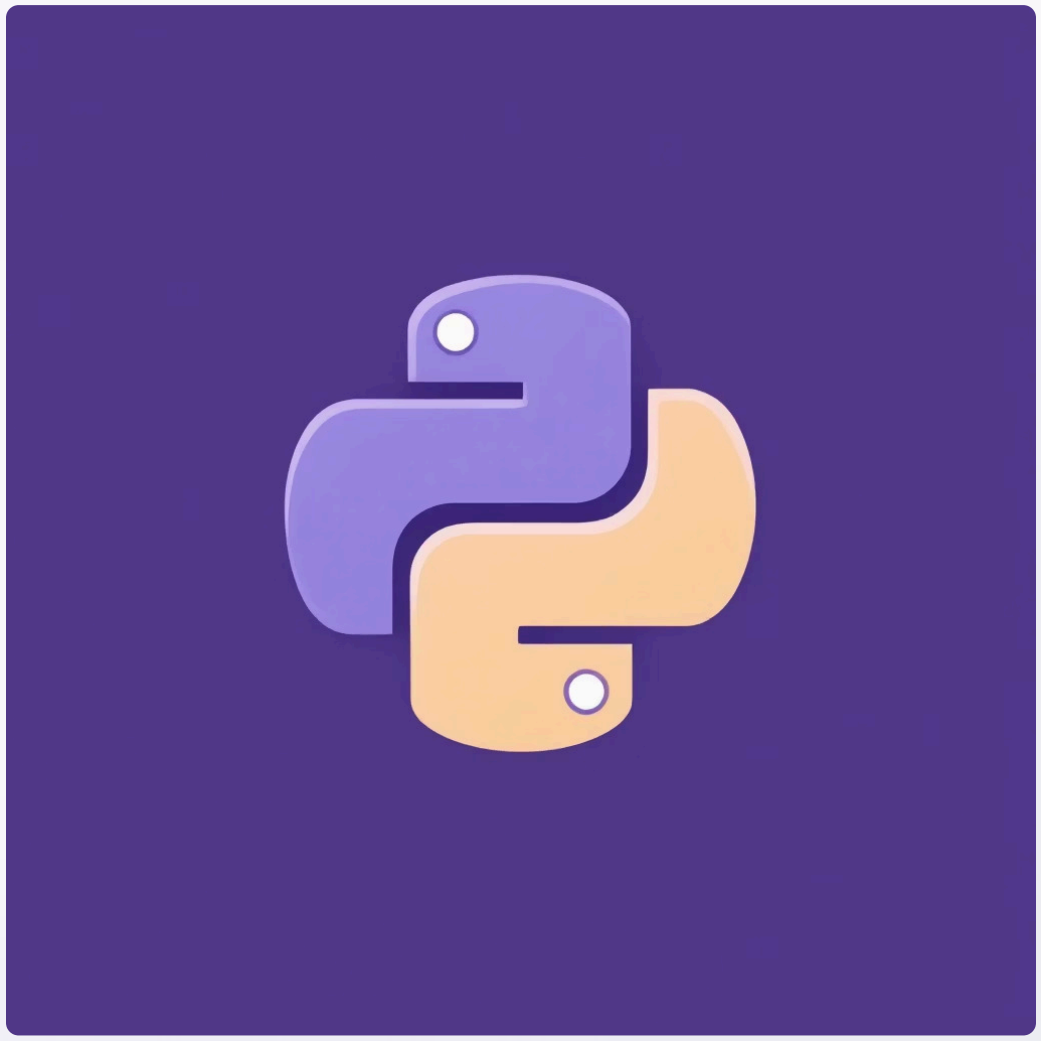


Introduction to Python Programming

What is Python?

Python is a **high-level, interpreted, general-purpose programming language** that emphasizes code simplicity and readability. Created by Guido van Rossum in 1991, it was named after the British comedy group Monty Python, reflecting its philosophy of making programming fun and accessible.

Python supports multiple programming paradigms including **procedural**, **object-oriented**, and **functional programming**, making it incredibly versatile for different problem-solving approaches.



Artificial Intelligence

Machine learning, deep learning, neural networks

Web Development

Django, Flask, FastAPI frameworks

Data Science

NumPy, Pandas, data analysis

Automation

Scripts, testing, DevOps tools

Using the Interactive Shell

The Python interactive shell (also called REPL - Read-Eval-Print Loop) is a powerful tool that allows developers to execute Python commands immediately and see results in real-time. It's perfect for testing code snippets, exploring libraries, and learning Python syntax.

Instant Execution

Type `>>> 2 + 2` and get `4` immediately. No need to write complete programs for simple tests.

Expression Evaluation

Expressions are pieces of code that Python evaluates to produce a value. Try `>>> 10 * 5 + 3` to see operator precedence in action.

Quick Learning

Perfect for beginners to experiment with syntax, test functions, and understand how Python interprets different commands without running full scripts.

Common Operators: Addition (+), Subtraction (-), Multiplication (*), Division (/), Modulus (%), Floor Division (//), Exponentiation (**)

Variables, Data Types, and Expressions

Variables in Python

Variables are containers that store data values. Unlike some languages, Python doesn't require explicit type declaration — it uses **dynamic typing**. Variable names must start with a letter or underscore, followed by letters, numbers, or underscores.

Naming Rules

- Cannot use Python keywords (if, for, while, etc.)
- Case-sensitive: `Age` and `age` are different
- Use descriptive names: `student_count` not `sc`

1

Integer (int)

Whole numbers without decimals

```
age = 19
count = -5
```

10

Float

Numbers with decimal points

```
price = 99.99
pi = 3.14159
```

String (str)

Text enclosed in quotes

```
name = "Abhishek"
msg = 'Hello'
```

Boolean (bool)

True or False values

```
is_student = True
has_passed = False
```

Type Conversion: Use `int()`, `float()`, and `str()` functions to convert between data types. Example: `int("42")` converts string to integer 42.

Statements and Comments

Statements

Instructions that Python executes to perform actions. Each statement tells Python to do something specific.

Comments

Human-readable notes that Python ignores. Essential for code documentation and team collaboration.

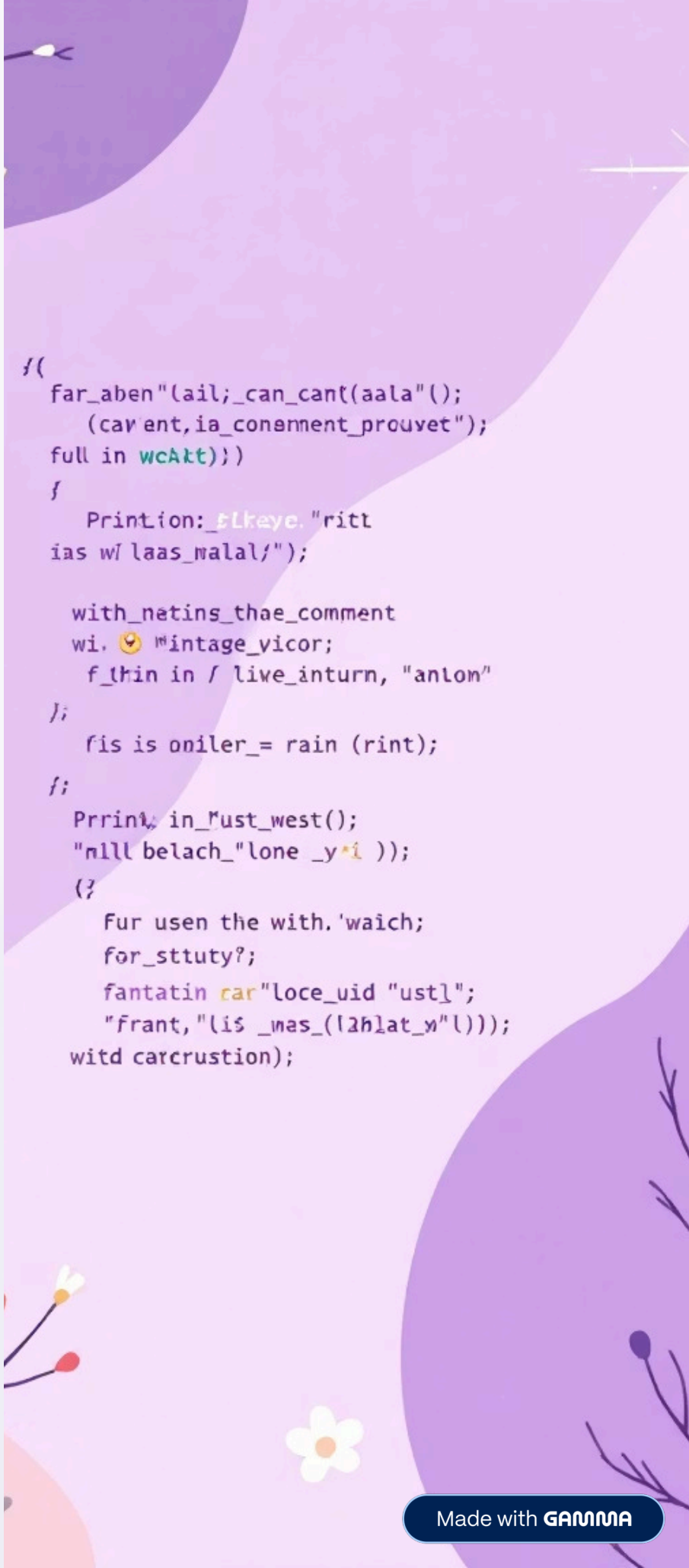
Common Statements

```
# Assignment statement
x = 10

# Print statement
print("Hello, world!")

# Multiple assignments
a, b, c = 1, 2, 3

# Input statement
name = input("Enter name: ")
```

Comment Best Practices

- **Single-line comments:** Use `#` symbol
- **Multi-line comments:** Use triple quotes `"""`
- Explain *why*, not *what* (code shows what)
- Keep comments concise and relevant
- Update comments when code changes

Pro Tip: Good commenting improves code maintainability and helps other developers (and future you) understand your logic quickly.



Flow Control: Boolean Values and Operators

Boolean logic is the foundation of decision-making in programming. It allows programs to evaluate conditions and choose different paths of execution based on whether conditions are true or false.

Boolean Values

Only two possible values: `True` and `False` (note the capitalization). Used to represent yes/no, on/off states.

Comparison Operators

`==` equal to, `!=` not equal, `<` less than, `>` greater than, `<=` less/equal, `>=` greater/equal

Boolean Operators

`and` (both true), `or` (at least one true), `not` (inverts boolean value)

Example: Simple Conditions

```
age = 20
is_adult = age >= 18  # True

score = 85
passed = score > 60  # True
```

Example: Complex Conditions

```
age = 20
has_id = True
can_enter = age >= 18 and has_id

is_weekend = True
is_holiday = False
day_off = is_weekend or is_holiday
```

Conditional Statements

Conditional statements allow programs to make decisions and execute different code blocks based on whether conditions evaluate to True or False. They're essential for creating dynamic, responsive programs.

1

if Statement

Executes code block only if the condition is True. If condition is False, the code block is skipped entirely.

```
if temperature > 30:
    print("It's hot outside!")
```

if-else Statement

Provides an alternative path when the condition is False. One of the two blocks will always execute.

```
if age >= 18:
    print("You can vote")
else:
    print("Too young to vote")
```

if-elif-else Statement

Handles multiple conditions sequentially. Python checks each condition in order and executes the first True block.

```
if score >= 90:
    grade = 'A'
elif score >= 80:
    grade = 'B'
elif score >= 70:
    grade = 'C'
else:
    grade = 'F'
```

Remember: Indentation is crucial in Python! All code within a conditional block must be indented consistently (typically 4 spaces). Python uses indentation to determine code structure.