ChatGPT

Module 1: Programming for Everybody (Getting Started with Python)

Module Overview: This module introduces Python 3 as a beginner-friendly, high-level programming language. Python (created by Guido van Rossum in 1991) emphasizes readable, English-like syntax, making it a great first language [1] [2]. It's open-source and cross-platform (runs on Windows, Linux, Mac, etc.) [3]. Python's design philosophy ("readability counts" [4]) means that even complex programs can be written simply. The module has no prerequisites and focuses on basic constructs (variables, expressions, conditionals, loops) so that anyone with moderate computer experience can quickly begin writing Python code.

Python as a Language

Image: The Python programming language logo. Python is an **open-source, general-purpose programming language** known for its easy-to-read code [1]. It is an **interpreted, high-level** language (meaning code is run by an interpreter rather than compiled, and syntax is close to human language) [2]. Python supports multiple paradigms (object-oriented, procedural, functional) with simple syntax and uses indentation to delimit code blocks (no braces needed) [2]. For example, printing a message in Python is just one line:

```
print("Hello, world!")   # outputs Hello, world! to the screen
```

Python comes with a large **"batteries included" standard library** and thousands of third-party packages (for web development, data analysis, machine learning, etc.), making it very versatile [1]. Its **dynamic typing** means you don't need to declare variable types – the interpreter figures them out at runtime [5] [6]. Overall, Python's focus on clarity and productivity lets beginners write useful programs quickly [1].

Eben Upton and the Raspberry Pi

Eben Upton is a British computer scientist (and Broadcom engineer) who co-founded the **Raspberry Pi Foundation**. Upton led the project to create the Raspberry Pi – a credit-card sized, low-cost computer – to give young people affordable hardware for learning programming [7] [8]. In 2008 Upton and colleagues set out to **rekindle interest in computer science** by designing an ultra-cheap PC that could run real software [8]. The result was the first Raspberry Pi Model B (released in 2012 for \$35) [9], which quickly became very popular (over 68 million units sold by 2025 [10]). Today Raspberry Pi boards range from simple microcontrollers (Pi Pico) to powerful mini-PCs (Pi 5 with up to 16 GB RAM) [11]. All Pis run Linux and often use Python as the main language for education and projects.

Raspberry Pi was **built for coding education**, so it comes with Python support out of the box. In fact, the name "Raspberry Pi" nods to the Python language (the developers named it after the fruit and the Python programming language) [12]. For example, you can control hardware attached to the Pi using Python's GPIO

libraries. Here's a simple Python snippet that blinks an LED on a GPIO pin:

```
import RPi.GPIO as GPIO
from time import sleep

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)
GPIO.setup(8, GPIO.OUT, initial=GPIO.LOW)

while True:
    GPIO.output(8, GPIO.HIGH)   # turn on LED
    sleep(1)
    GPIO.output(8, GPIO.LOW)    # turn off LED
    sleep(1)
```

This script repeatedly toggles pin 8 high/low to blink an LED every second. The Raspberry Pi's `RPi.GPIO` Python library is used to access the hardware pins. In general, on a Pi you'll use familiar Python syntax and libraries to write real-world programs (sensors, robotics, web servers, etc.), making Python and Raspberry Pi a powerful combination for learning and prototyping [13] [12].

Variables and Expressions

In Python, a **variable** is a name (symbol) that refers to a value stored in memory [5]. You create a variable by assignment using the `=` operator. For example:

```
x = 5         # creates a variable x referring to the integer 5
name = "Bob"  # creates a variable name referring to the string "Bob"
```

Here `x` and `name` are variables. Python figures out their types automatically (no need to declare "int" or "string") [5] [6]. Variable names can contain letters, digits and underscores but must start with a letter or underscore (not a digit) [14]. They are case-sensitive (`Data` and `data` are different) [14].

An **expression** is any combination of values, variables, and operators that Python can evaluate to produce a value [15]. For example: - **Numeric expressions:** `3 + 4 * 2` combines numbers and arithmetic operators. Python evaluates multiplication before addition, giving `3 + 8 = 11`.
- **String expressions:** `"Hi, " + name` concatenates strings, resulting in `"Hi, Bob"` if `name` is `"Bob"`.
- **Boolean expressions:** `x > 10` compares values using `>` (greater than), returning `True` or `False`.
Every expression produces a single result. You can assign expressions to variables, e.g.:

```
x = 5
y = 3
```

```
sum_xy = x + y   # expression x+y yields 8
print(sum_xy)    # prints 8
```

Here `x+y` is an arithmetic expression whose value is stored in `sum_xy`. You can also use built-in functions and operators: `result = (10 - 3) * 2` or `initial = name[0]` to get the first letter of `name`. Learning to combine variables and expressions lets you perform calculations and manipulate data in your code [15] [16].

Conditional Code

Conditional code allows a Python program to make decisions. The main constructs are `if`, `elif` (else-if), and `else` statements. An `if` statement evaluates a boolean condition (True/False) and runs an indented block of code only if the condition is true [17]. For example:

```
age = 20
if age >= 18:
    print("You are an adult")
```

This checks whether `age >= 18`. If that expression is True, it executes the `print` line; otherwise it skips it. The condition `age >= 18` is a Boolean expression using the `>=` (greater-or-equal) operator. Note that `==` **(double equals)** is used for equality comparison (e.g. `if x == 5:`), whereas a single `=` is assignment [18].

Proper **indentation** is crucial: the code under `if` must be indented (typically by 4 spaces). Only indented lines belong to the `if`. Unindented code runs afterward. For example:

```
num = 4
if num > 0:
    print("Positive")
print("Done")  # this runs regardless of the condition
```

If `num > 0`, it prints `"Positive"`; either way it then prints `"Done"`. This illustrates how conditionals change the flow of execution [17].

Conditional Statements

Python's `if`-`elif`-`else` chain handles multiple branches. The `elif` keyword (short for *else if*) lets you check additional conditions if previous ones were false. The optional `else` block runs when none of the earlier conditions were true [19]. For example:

```
score = 85
if score >= 90:
```

```
    grade = "A"
elif score >= 80:
    grade = "B"
elif score >= 70:
    grade = "C"
else:
    grade = "F"
print("Grade:", grade)
```

Here Python checks each condition in order. Since 85 ≥ 80, it sets `grade = "B"`. If none of the `if` or `elif` tests succeed, the `else` code runs (assigning `"F"` in this case). In general, use `if` for the first test, use one or more `elif` for subsequent tests, and optionally `else` for a final catch-all case [19].

You can also nest conditionals (an `if` inside another `if`) for more complex logic. The key idea is that each `if`, `elif` or `else` is followed by a condition (except `else`) and a colon, and its block is indented [19]. Conditional statements let programs react differently: for example, printing different messages, executing different computations, or looping under different conditions based on user input or data.

Loops and Iteration

Loops repeat a block of code multiple times. Python has two main loop constructs: `for` **loops** and `while` **loops**.

- **For loops:** Used for iterating over sequences (lists, strings, ranges, etc.) [20]. The syntax is:

  ```
  for variable in sequence:
      # code block (body) to run for each item
  ```

  For example:

  ```
  for i in range(3):
      print(i)
  ```

  This loop sets `i` to 0, then 1, then 2 (the values in `range(3)`) and prints each. The body (the `print` line) is indented and executes once per item [20]. A `for` loop ends after it has processed all items. You can loop over any sequence, e.g. a list of strings or even characters in a string:

  ```
  for char in "Python":
      print(char)
  ```

  would print each letter in “Python” one per line. For-loops are great when you know how many iterations you need (or have a specific sequence of items).

- **While loops:** Used for repeating code as long as a condition remains True [21]. The syntax is:

```
while condition:
    # code block to repeat
```

For example:

```
count = 0
while count < 3:
    print(count)
    count = count + 1
```

Here the loop checks `count < 3`. As long as that is True, it prints `count` and then increments it. When `count` becomes 3, the condition is False and the loop stops [21]. It is important to update variables used in the condition (here `count`) so that the loop eventually terminates (otherwise you get an infinite loop) [21] [22].

Loops often use `break` (to exit immediately) or `continue` (to skip to the next iteration), but basic loops simply run their indented body repeatedly. In summary, `for` loops iterate over fixed sequences (nice for arrays or known ranges), whereas `while` loops continue as long as a condition holds (useful when the number of iterations isn't predetermined) [21]. These constructs let you perform tasks like summing numbers, processing list items, or repeatedly checking user input.

Sources: The above descriptions draw on Python reference materials and tutorials [1] [2] [7] [8] [12] [5] [15] [14] [19] [17] [20] [21].

---

[1] Python | Innovate Labs
https://innovatelabs.business.uconn.edu/tech/python/

[2] [3] [4] Python - Overview
https://www.tutorialspoint.com/python/python_overview.htm

[5] [6] [16] Variables in Python: Usage and Best Practices – Real Python
https://realpython.com/python-variables/

[7] Eben Upton CBE - Archives of IT
https://archivesit.org.uk/interviews/eben-upton-cbe/

[8] [9] [10] [11] [12] [13] Raspberry Pi - Wikipedia
https://en.wikipedia.org/wiki/Raspberry_Pi

[14] Python - Variables
https://www.tutorialspoint.com/python/python_variables.htm

[15] Expressions in Python - GeeksforGeeks
https://www.geeksforgeeks.org/computer-science-fundamentals/expressions-in-python/

17 18 cs.stanford.edu
https://cs.stanford.edu/people/nick/py/python-if.html

19 How to Use Conditional Statements in Python – Examples of if, else, and elif
https://www.freecodecamp.org/news/how-to-use-conditional-statements-if-else-elif-in-python/

20 Python for Loop (With Examples)
https://www.programiz.com/python-programming/for-loop

21 22 Python while Loop (With Examples)
https://www.programiz.com/python-programming/while-loop