



**Gurukul Technicals Institute**

Computer education · Robotics · Science projects · Delhi

**FREE EBOOK**

PROGRAMMING AND WEB

# Python for Beginners

Learn to program with Python 3, one small program at a time

---

## INSIDE THIS BOOK

- Getting ready: Python 3 and IDLE
- Talking to the user: print, input and variables
- Numbers and arithmetic
- Working with strings
- Making decisions with if, elif and else
- ...and 6 more chapters, practice questions and a glossary

---

**Beginner**

School and college students starting programming

**engineergurukul.com**

+91 97116 99511

# Python for Beginners

A gentle first course in programming with Python 3. You will install Python and IDLE, then learn variables, numbers, strings, decisions, loops, lists, dictionaries, functions, modules, files and error handling by writing fifteen small working programs, from a simple interest calculator to a guessing game and a class marks report.

|                |                                                              |
|----------------|--------------------------------------------------------------|
| WHO IT IS FOR  | School and college students starting programming             |
| LEVEL          | Beginner                                                     |
| SUBJECT AREA   | Programming and web                                          |
| WHAT IS INSIDE | 11 chapters, practice questions with answers, and a glossary |

## CONTENTS

|    |                                                 |    |
|----|-------------------------------------------------|----|
| 1  | Getting ready: Python 3 and IDLE                | 4  |
| 2  | Talking to the user: print, input and variables | 5  |
| 3  | Numbers and arithmetic                          | 7  |
| 4  | Working with strings                            | 9  |
| 5  | Making decisions with if, elif and else         | 10 |
| 6  | Repeating with while and for loops              | 12 |
| 7  | Lists, tuples and dictionaries                  | 15 |
| 8  | Writing your own functions                      | 17 |
| 9  | Modules: math and random                        | 19 |
| 10 | Files, errors and try/except                    | 20 |
| 11 | Project: a class marks report                   | 23 |

|                           |           |
|---------------------------|-----------|
| <b>Practice questions</b> | <b>24</b> |
| <b>Answers</b>            | <b>25</b> |
| <b>Glossary</b>           | <b>26</b> |

Free ebook: you may read, print and share it for learning; it may not be sold or changed. © 2026 Gurukul Technicals Institute. First edition, October 2026. Contact details are on the back cover.

# Getting ready: Python 3 and IDLE

A **program** is a list of instructions that a computer follows, one after another. A **programming language** is a special language for writing those instructions. Python is one of the easiest programming languages to learn. Its instructions look a lot like short English sentences, so you can read a Python program and guess what it does.

Python is free. It is used to build websites, study data, control robots and teach programming in schools and colleges. This book uses **Python 3**, the current version. Very old books and videos may show Python 2, which is no longer supported, so check the version before you copy from them.

## Installing Python on a computer

1. On a Windows computer, open a web browser and go to [python.org](https://python.org). Open the Downloads section and download Python for Windows.
2. Run the file you downloaded and follow the steps on screen.
3. If the installer shows a box about adding Python to **PATH**, tick it. PATH is a list of folders where Windows looks for programs, so this lets you start Python from anywhere.
4. When it finishes, open the Start menu and search for **IDLE**.

On a Mac, the installer from [python.org](https://python.org) also installs IDLE. On Linux, Python 3 is usually already present, but IDLE may need to be added from the software centre. On a school or office computer, ask your teacher or the system administrator before installing anything.

### TIP

To check that Python works, open Command Prompt and type `python --version` (on Mac or Linux, type `python3 --version`). It should print a line such as Python 3 followed by more numbers. If the Start menu shows no IDLE, typing `python -m idlelib` in Command Prompt also opens it.

## Meeting IDLE

**IDLE** is a small program that comes with Python. You type and run your Python code in it. It has two kinds of window:

| Window | What it looks like                         | What it is for                    |
|--------|--------------------------------------------|-----------------------------------|
| Shell  | Lines start with <code>&gt;&gt;&gt;</code> | Trying one line at a time         |
| Editor | A blank page                               | Writing and saving whole programs |

Type this in the Shell after the `>>>` and press Enter:

```
>>> print("Namaste, world!")
Namaste, world!
```

The computer did exactly what you asked. Now write your first saved program:

1. In IDLE, click **File**, then **New File**. A blank editor window opens.
2. Type `print("Namaste, world!")`.
3. Click **File**, then **Save**. Make a folder called `python-practice` and save the file as `hello.py`. The ending `.py` tells the computer that this is a Python file.
4. Click **Run**, then **Run Module**, or press F5. The answer appears in the Shell window.

#### REMEMBER

Python is **case sensitive**. It treats capital and small letters as different. `print` works, but `Print` and `PRINT` give an error.

#### DID YOU KNOW

Python is not named after the snake. Its creator, Guido van Rossum, named it after a British comedy show called Monty Python's Flying Circus.

## CHAPTER 2

# Talking to the user: print, input and variables

A useful program talks to the person using it. It shows messages with `print()` and asks questions with `input()`.

## Showing messages with print

You can give `print()` several items separated by commas. Python puts a space between them.

#### PYTHON

```
print("Rohan scored", 45, "runs")
print()
print("Match over")
```

This prints `Rohan scored 45 runs`, then a blank line, then `Match over`. Text inside quotes is called a **string**. Numbers do not need quotes.

A line that starts with `#` is a **comment**. Python ignores it. Comments are notes for people who read your code.

## Storing values in variables

A **variable** is a name that holds a value, like a labelled box. You put a value in it with the `=` sign.

### PYTHON

```
student = "Ananya"
age = 14
height = 1.52
is_monitor = True
print(student, "is", age, "years old")
```

Here `=` does not mean "equals" as in maths. It means "store the value on the right in the name on the left". You can change a variable later with another `=`.

Rules for variable names:

- Use letters, digits and the underscore (`_`). No spaces.
- Do not start with a digit. `class2` is fine; `2class` is not.
- Do not use Python's special words, such as `if`, `for` and `while`, and avoid names Python already uses, such as `print` and `input`.
- Choose names that explain themselves: `total_marks` is better than `tm`.

## Types of values

Every value has a **type**. The four basic types are:

| Type           | Name in Python     | Examples      |
|----------------|--------------------|---------------|
| Whole number   | <code>int</code>   | 7, 0, -25     |
| Decimal number | <code>float</code> | 3.5, 99.9     |
| Text           | <code>str</code>   | "Delhi", "42" |
| Yes or no      | <code>bool</code>  | True, False   |

The `type()` function tells you the type of any value. In the Shell, `type(3.5)` shows `<class 'float'>`.

## Asking questions with input

`input()` shows a question, waits for the user to type an answer, and gives back what they typed.

```
# Program 1: A friendly greeting
name = input("What is your name? ")
city = input("Which city do you live in? ")
print("Namaste,", name + "!")
print("It is nice to meet someone from", city + ".")
```

If Riya from Jaipur runs it, she sees `Namaste, Riya!` and `It is nice to meet someone from Jaipur.` The `+` joins two strings into one, so no space appears before the `!`.

#### REMEMBER

`input()` always gives back a string, even when the user types a number. To do maths with it, change it with `int()` for whole numbers or `float()` for decimals, for example `age = int(input("Age: "))`.

## CHAPTER 3

# Numbers and arithmetic

Python is a very good calculator. These are its **arithmetic operators**, the symbols for maths:

| Operator | Meaning                     | Example             | Result             |
|----------|-----------------------------|---------------------|--------------------|
| +        | Add                         | <code>8 + 3</code>  | 11                 |
| -        | Subtract                    | <code>8 - 3</code>  | 5                  |
| *        | Multiply                    | <code>8 * 3</code>  | 24                 |
| /        | Divide                      | <code>8 / 3</code>  | 2.6666666666666665 |
| //       | Divide, keep the whole part | <code>8 // 3</code> | 2                  |
| %        | Remainder                   | <code>8 % 3</code>  | 2                  |
| **       | Power                       | <code>8 ** 2</code> | 64                 |

Two of these surprise beginners. Normal division `/` always gives a float, so `6 / 2` gives `3.0`, not `3`. And `%` gives the **remainder** after division. It is useful for checking even numbers: if `n % 2` is 0, then `n` is even.

## Order of working

Python follows the same order you learnt as BODMAS. Brackets come first, then powers, then multiply and divide, then add and subtract. So `10 + 5 * 2` is 20, while `(10 + 5) * 2` is 30. When in doubt, add brackets. They make your meaning clear.

## Updating a variable

To add to a variable, you can write `score = score + 4`. Python has a short form: `score += 4`. There are also `-=`, `*=` and `/=`.

## Neat decimals with f-strings

An **f-string** is a string with the letter `f` before the opening quote. Inside it, anything in curly brackets `{}` is replaced by its value. Writing `:.2f` after a value shows exactly two decimal places, which is perfect for rupees and paise.

### PYTHON

```
# Program 2: Simple interest
principal = float(input("Principal amount in rupees: "))
rate = float(input("Rate of interest per year (%): "))
years = float(input("Time in years: "))

interest = principal * rate * years / 100
amount = principal + interest

print(f"Simple interest: ₹{interest:.2f}")
print(f"Total amount: ₹{amount:.2f}")
```

### EXAMPLE

Mr Verma of Lucknow lends ! 25,000 to a friend at an example rate of 7.5% per year for 2 years. The program prints Simple interest: ₹3750.00 and Total amount: ₹28750.00. Check it by hand:  $25000 \times 7.5 \times 2 \div 100 = 3750$ .

### PYTHON

```
# Program 3: Celsius to Fahrenheit
celsius = float(input("Temperature in Celsius: "))
fahrenheit = celsius * 9 / 5 + 32
print(f"{celsius} °C is {fahrenheit:.1f} °F")
```

Type 37 and it prints 37.0 °C is 98.6 °F, which is about normal body temperature.

### DID YOU KNOW

Type `0.1 + 0.2` in the Shell and Python shows `0.30000000000000004`. The computer stores decimals in binary (only 0s and 1s), and some decimals cannot be stored exactly. This is normal in every language. Use `:.2f` or `round(value, 2)` when you display money.

# Working with strings

A **string** is any text: a name, a sentence, a PIN code or a whole paragraph. You can write it in single quotes `'like this'` or double quotes `"like this"`. Use double quotes when the text has an apostrophe: `"Priya's book"`.

## Counting and picking letters

`len()` gives the length of a string, counting spaces too. `len("New Delhi")` is 9.

Each character has a position number called an **index**. Counting starts from 0, not 1. A negative index counts from the end.

| Character      | P  | Y  | T  | H  | O  | N  |
|----------------|----|----|----|----|----|----|
| Index          | 0  | 1  | 2  | 3  | 4  | 5  |
| Negative index | -6 | -5 | -4 | -3 | -2 | -1 |

So if `word = "PYTHON"`, then `word[0]` is `"P"` and `word[-1]` is `"N"`.

A **slice** takes a part of a string. `word[1:4]` starts at index 1 and stops before index 4, giving `"YTH"`. Leave out a number to go to the start or end: `word[:2]` is `"PY"` and `word[3:]` is `"HON"`. The slice `word[::-1]` reads the string backwards, giving `"NOHTYP"`.

## String methods

A **method** is a function that belongs to a value. You call it with a dot after the value.

| Method                                      | What it does                        | <code>"Mumbai Local "</code><br>becomes |
|---------------------------------------------|-------------------------------------|-----------------------------------------|
| <code>.upper()</code>                       | All capitals                        | <code>"MUMBAI LOCAL "</code>            |
| <code>.lower()</code>                       | All small letters                   | <code>"mumbai local "</code>            |
| <code>.strip()</code>                       | Removes spaces at both ends         | <code>"Mumbai Local"</code>             |
| <code>.replace("Local",<br/>"Metro")</code> | Swaps one piece of text for another | <code>"Mumbai Metro "</code>            |
| <code>.split()</code>                       | Breaks into a list of words         | <code>["Mumbai", "Local"]</code>        |
| <code>.count("a")</code>                    | Counts a letter or piece            | <code>2</code>                          |

Methods do not change the original string. Strings cannot be changed once made, so each method gives back a new string. Store it in a variable if you want to keep it.

You can also join strings with `+`, repeat them with `*` (so `"-" * 20` draws a line of 20 dashes), and test whether one is inside another with `in`: `"Delhi" in "New Delhi"` is `True`.

#### PYTHON

```
# Program 4: Username maker
first = input("First name: ").strip()
year = input("Year of birth: ").strip()
username = first.lower() + year[-2:]
print("Your username is", username)
print("It has", len(username), "characters.")
```

#### EXAMPLE

Kabir types Kabir and 2011. The program takes kabir and the last two characters of the year, 11, and prints Your username is kabir11 and It has 7 characters.

#### TRY THIS

Change Program 4 so the username uses the first three letters of the name only. Hint: use a slice.

## CHAPTER 5

# Making decisions with if, elif and else

Programs become clever when they can choose. The `if` statement runs some lines only when a **condition** is true. A condition is a question whose answer is `True` or `False`.

## Comparing values

| Operator           | Meaning                  | Example                  | Answer |
|--------------------|--------------------------|--------------------------|--------|
| <code>==</code>    | Is equal to              | <code>5 == 5</code>      | True   |
| <code>!=</code>    | Is not equal to          | <code>5 != 3</code>      | True   |
| <code>&gt;</code>  | Is greater than          | <code>7 &gt; 9</code>    | False  |
| <code>&lt;</code>  | Is less than             | <code>2 &lt; 9</code>    | True   |
| <code>&gt;=</code> | Greater than or equal to | <code>40 &gt;= 40</code> | True   |
| <code>&lt;=</code> | Less than or equal to    | <code>39 &lt;= 40</code> | True   |

Notice the double `==`. A single `=` stores a value; a double `==` compares two values. Mixing them up is the most common beginner mistake.

## Writing an if statement

PYTHON

```
temperature = 41
if temperature > 40:
    print("Very hot. Drink plenty of water.")
else:
    print("Have a good day.")
```

Two things matter here. The `if` line ends with a colon (:). The lines that belong to it are **indented**: pushed right by four spaces. IDLE adds the spaces for you after a colon. Python uses indentation to know which lines belong to the `if`, so it must be the same in every line of the block.

Use `elif` (short for "else if") to check more conditions one after another. Python checks from the top and runs only the first block whose condition is true.

PYTHON

```
# Program 5: Grade calculator
marks = float(input("Enter marks out of 100: "))

if marks < 0 or marks > 100:
    print("Please enter marks between 0 and 100.")
elif marks >= 90:
    print("Grade A")
elif marks >= 75:
    print("Grade B")
elif marks >= 60:
    print("Grade C")
elif marks >= 40:
    print("Grade D")
else:
    print("Grade E: keep practising")
```

### NOTE

These grade bands are only an example. Your school or board may use different ones.

## Joining conditions

- `and` is true only when both sides are true.
- `or` is true when at least one side is true.
- `not` turns True into False and False into True.

#### PYTHON

```
# Program 6: Palindrome check
text = input("Enter a word or phrase: ")
cleaned = text.lower().replace(" ", "")
if cleaned == cleaned[::-1]:
    print(f'"{text}" is a palindrome.')
else:
    print(f'"{text}" is not a palindrome.')
```

A **palindrome** reads the same backwards and forwards. Try **Malayalam**, **Nitin** and **Never odd or even**. The program makes everything small and removes spaces before comparing.

#### PYTHON

```
# Program 7: Leap year check
year = int(input("Enter a year: "))
if (year % 4 == 0 and year % 100 != 0) or year % 400 == 0:
    print(year, "is a leap year.")
else:
    print(year, "is not a leap year.")
```

The rule: a year is a leap year if it divides by 4 but not by 100, or if it divides by 400. So 2024 and 2000 are leap years, but 1900 is not.

#### REMEMBER

Put a colon at the end of every **if**, **elif** and **else** line, and indent the lines under it.

## CHAPTER 6

# Repeating with while and for loops

A **loop** repeats lines of code. Imagine writing 100 **print()** lines to count to 100. A loop does it in three.

## The while loop

A **while** loop repeats as long as its condition is true.

#### PYTHON

```
count = 1
while count <= 5:
    print("Round", count)
    count += 1
print("Done")
```

It prints Round 1 to Round 5. Each time round, `count` grows by 1. When it reaches 6, the condition becomes false and the loop stops.

#### TIP

If you forget `count += 1`, the condition stays true for ever. This is an **infinite loop**. Press Ctrl+C to stop the program.

## The for loop and range

A `for` loop goes through a series of values one by one. `range()` makes a series of numbers:

| Code                          | Numbers it gives |
|-------------------------------|------------------|
| <code>range(5)</code>         | 0, 1, 2, 3, 4    |
| <code>range(1, 6)</code>      | 1, 2, 3, 4, 5    |
| <code>range(2, 11, 2)</code>  | 2, 4, 6, 8, 10   |
| <code>range(10, 0, -1)</code> | 10, 9, 8 ... 1   |

The stop number is never included. That is why `range(1, 11)` is used to count from 1 to 10.

#### PYTHON

```
# Program 8: Multiplication table
number = int(input("Which table do you want? "))
for i in range(1, 11):
    print(f"{number} x {i} = {number * i}")
```

A `for` loop can also go through each letter of a string: `for letter in "INDIA":` runs five times.

## Stopping early with break and continue

`break` leaves the loop at once. `continue` skips the rest of this round and goes on to the next one.

## PYTHON

```
# Program 9: Prime check
n = int(input("Enter a whole number: "))

if n < 2:
    print(n, "is not a prime number.")
else:
    is_prime = True
    d = 2
    while d * d <= n:
        if n % d == 0:
            is_prime = False
            break
        d += 1
    if is_prime:
        print(n, "is a prime number.")
    else:
        print(f"{n} is not a prime number. It divides by {d}.")
```

A **prime number** has exactly two factors: 1 and itself. The program tries each divisor `d` from 2 upwards. It only needs to go while `d * d` is not more than `n`. Any factor bigger than the square root of `n` comes in a pair with a factor smaller than it, which the loop would already have found. Type 97 and it says prime; type 91 and it says it divides by 7.

## PYTHON

```
# Program 10: Fibonacci series
count = int(input("How many Fibonacci numbers? "))
a, b = 0, 1
for i in range(count):
    print(a, end=" ")
    a, b = b, a + b
print()
```

In the **Fibonacci series**, each number is the sum of the two before it: 0 1 1 2 3 5 8 13 and so on. The line `a, b = b, a + b` updates both variables at once. Writing `end=" "` makes `print()` end with a space instead of starting a new line.

## DID YOU KNOW

Indian scholars such as Virahanka described this number pattern centuries before Fibonacci wrote about it in 1202. They found it while counting the rhythms of long and short syllables in poetry.

# Lists, tuples and dictionaries

So far each variable held one value. Real programs handle many values: the marks of a whole class, or every item on a shopping list. Python has three main ways to store a group of values.

## Lists

A **list** holds items in order, inside square brackets.

### PYTHON

```
fruits = ["mango", "banana", "guava"]
fruits.append("apple")      # add at the end
fruits.insert(0, "papaya")  # add at index 0
fruits.remove("banana")     # remove by value
print(fruits)               # ['papaya', 'mango', 'guava', 'apple']
print(len(fruits), fruits[1]) # 4 mango
```

Lists use index numbers just like strings, starting at 0. Unlike strings, lists can be changed. Some useful tools:

| Code                                | What it gives                        |
|-------------------------------------|--------------------------------------|
| <code>sum(marks)</code>             | Total of all numbers                 |
| <code>max(marks), min(marks)</code> | Largest and smallest                 |
| <code>sorted(marks)</code>          | A new sorted list                    |
| <code>marks.sort()</code>           | Sorts the list itself                |
| <code>marks.pop()</code>            | Removes and gives back the last item |
| <code>72 in marks</code>            | True if 72 is in the list            |

A **for** loop goes through a list easily:

### PYTHON

```
marks = [72, 88, 65, 91, 79]
for m in marks:
    print(m)
print("Average:", sum(marks) / len(marks)) # Average: 79.0
```

## Tuples

A **tuple** is like a list that cannot be changed. It uses round brackets: `days = ("Mon", "Tue", "Wed")`. Use a tuple for values that should stay fixed, such as the months of the year. You can **unpack** a tuple into separate variables: `city, pin = ("Patna", "800001")`.

## Dictionaries

A **dictionary** stores pairs. Each pair has a **key** and a **value**, like a word and its meaning in a real dictionary.

### PYTHON

```
phone = {"Amit": "98100 11111", "Neha": "98200 22222"}
phone["Zoya"] = "98300 33333"          # add a new pair
print(phone["Neha"])                   # 98200 22222
print(phone.get("Ravi", "Not found"))  # Not found
for name, number in phone.items():
    print(name, number)
```

You look up a value by its key, not by position. Asking for a key that does not exist, such as `phone["Ravi"]`, causes an error. The `.get()` method is safer: it gives back a default value instead.

### PYTHON

```
# Program 11: Word count
text = input("Type a sentence: ").lower()
words = text.split()
counts = {}
for word in words:
    word = word.strip(".,!?")
    counts[word] = counts.get(word, 0) + 1

print("Total words:", len(words))
for word, n in counts.items():
    print(word, "-", n)
```

For `The cat saw the dog. The dog ran!` it prints a total of 8 words, then `the - 3`, `cat - 1`, `saw - 1`, `dog - 2`, `ran - 1`.

### REMEMBER

Square brackets `[ ]` make a list, round brackets `( )` make a tuple, and curly brackets `{ }` with key: value pairs make a dictionary.

# Writing your own functions

You have used Python's own functions: `print()`, `input()`, `len()`. A **function** is a named block of code that does one job. You can write your own with the word `def` (short for define).

## PYTHON

```
def greet(name):  
    print("Hello,", name)  
  
greet("Farhan")  
greet("Meenakshi")
```

The first two lines **define** the function. They do nothing yet. The last two lines **call** it, which means run it. `name` is a **parameter**: a variable that receives the value you pass in. The value you pass, such as `"Farhan"`, is called an **argument**.

## Giving back an answer with return

A function can send a value back to the line that called it, using `return`.

## PYTHON

```
def rectangle_area(length, width):  
    return length * width  
  
room = rectangle_area(12, 10)  
print("Room area:", room, "square feet") # 120
```

When Python reaches `return`, it leaves the function at once and hands back the value.

A parameter can have a **default value**, used when no argument is given:

## PYTHON

```
def price_with_discount(price, discount=10):  
    return price - price * discount / 100  
  
print(price_with_discount(500)) # 450.0  
print(price_with_discount(500, 20)) # 400.0
```

Variables made inside a function are **local**. They exist only while the function runs, and the rest of the program cannot see them.

## Why functions help

- You write the code once and use it many times.
- Each function has one clear job, so the program is easier to read.

- When something goes wrong, you fix it in one place.

#### PYTHON

```
# Program 12: Calculator
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b

def divide(a, b):
    return a / b

while True:
    print("1 Add  2 Subtract  3 Multiply  4 Divide  5 Quit")
    choice = input("Choose 1 to 5: ")
    if choice == "5":
        print("Goodbye!")
        break
    if choice not in ("1", "2", "3", "4"):
        print("Please choose a number from 1 to 5.")
        continue
    x = float(input("First number: "))
    y = float(input("Second number: "))
    if choice == "1":
        print("Answer:", add(x, y))
    elif choice == "2":
        print("Answer:", subtract(x, y))
    elif choice == "3":
        print("Answer:", multiply(x, y))
    elif y == 0:
        print("You cannot divide by zero.")
    else:
        print("Answer:", divide(x, y))
```

`while True:` makes a loop that runs until a `break`. Here the user chooses 5 to quit. The check `y == 0` comes before division, because dividing by zero stops a program with an error.

#### TRY THIS

Add a sixth choice to the calculator that finds the remainder using `%`.

# Modules: math and random

A **module** is a file of ready-made functions that someone else has written. Python comes with many modules. You bring one into your program with `import`, and then use its functions with a dot.

## The math module

PYTHON

```
import math

print(math.sqrt(144))      # 12.0 square root
print(math.pi)             # 3.141592653589793
print(math.ceil(4.2))      # 5 round up
print(math.floor(4.8))     # 4 round down
print(math.factorial(5))   # 120 5 x 4 x 3 x 2 x 1

radius = 7
area = math.pi * radius ** 2
print(f"Area of the circle: {area:.2f} sq cm") # 153.94
```

EXAMPLE

A caterer in Indore needs one tray for every 12 guests. For 150 guests,  $150 / 12$  is 12.5. You cannot hire half a tray, so `math.ceil(150 / 12)` gives 13 trays.

You can also import just one function: `from math import sqrt`. Then you write `sqrt(81)` without `math.` in front.

## The random module

The **random** module picks values by chance, like rolling a dice.

| Function                           | What it does                              | Example result      |
|------------------------------------|-------------------------------------------|---------------------|
| <code>random.randint(1, 6)</code>  | A whole number from 1 to 6, both included | 4                   |
| <code>random.choice(names)</code>  | One item from a list                      | "Pooja"             |
| <code>random.shuffle(cards)</code> | Mixes up a list in place                  | (list is reordered) |
| <code>random.random()</code>       | A decimal from 0 up to (not including) 1  | 0.7312...           |

## PYTHON

```
import random

names = ["Pooja", "Arjun", "Sana", "Vikram"]
print("Today's class monitor:", random.choice(names))
print("Dice shows:", random.randint(1, 6))
```

Run it a few times. The answers change each time.

## PYTHON

```
# Program 13: Number guessing game
import random

secret = random.randint(1, 50)
tries = 0
print("I am thinking of a number from 1 to 50.")

while True:
    guess = int(input("Your guess: "))
    tries += 1
    if guess < secret:
        print("Too low. Try a bigger number.")
    elif guess > secret:
        print("Too high. Try a smaller number.")
    else:
        print(f"Correct! You found it in {tries} tries.")
        break
```

### TIP

The smartest way to play is to guess the middle number each time: 25, then 12 or 37, and so on. Each guess cuts the possible numbers in half, so you never need more than six guesses for 1 to 50.

### TRY THIS

Give the player only 7 tries. After 7 wrong guesses, print the secret number and end the game.

## CHAPTER 10

# Files, errors and try/except

When a program ends, its variables disappear. To keep information for next time, a program must save it in a **file** on the disk.

## Writing and reading text files

`open()` opens a file. The second value is the **mode**:

| Mode | Meaning                                                     |
|------|-------------------------------------------------------------|
| "r"  | Read. The file must already exist. This is the default.     |
| "w"  | Write. Makes a new file, or wipes an old one clean first.   |
| "a"  | Append. Adds to the end of the file, keeping what is there. |

Always open files with a `with` block. It closes the file for you when the block ends, even if something goes wrong.

### PYTHON

```
with open("friends.txt", "w", encoding="utf-8") as f:
    f.write("Asha\n")
    f.write("Bilal\n")

with open("friends.txt", encoding="utf-8") as f:
    for line in f:
        print(line.strip())
```

`\n` means "new line". `.strip()` removes it again when reading. `encoding="utf-8"` lets the file hold any language, including Hindi and the ! sign.

### SAFETY

Mode "w" wipes the old contents of a file without asking. Use "a" when you want to keep what is already saved, and keep a copy of any file that matters.

## When things go wrong

An **error** stops a program and prints a message called a **traceback**. Read its last line first: it names the error and gives a short reason.

| Error             | Common cause                                       |
|-------------------|----------------------------------------------------|
| SyntaxError       | A missing colon, bracket or quote                  |
| IndentationError  | Lines in a block not lined up                      |
| NameError         | A misspelt variable or function name               |
| TypeError         | Mixing types, such as <code>"Age: " + 14</code>    |
| ValueError        | <code>int("abc")</code> : the text is not a number |
| ZeroDivisionError | Dividing by zero                                   |

| Error             | Common cause                                   |
|-------------------|------------------------------------------------|
| IndexError        | Asking for a list position that does not exist |
| KeyError          | Asking a dictionary for a missing key          |
| FileNotFoundError | Opening a file that is not there               |

## Catching errors with try and except

Some errors are not your fault: the user types "ten" instead of 10, or a file is missing. Use `try` and `except` to handle them politely. Python tries the code under `try`. If an error happens, it jumps to the matching `except` instead of stopping. Code under `else` runs only if there was no error.

### PYTHON

```
# Program 14: Expense diary
FILE = "expenses.txt"

item = input("What did you spend on? ")
try:
    amount = float(input("How much in rupees? "))
except ValueError:
    print("That is not a number. Nothing was saved.")
else:
    with open(FILE, "a", encoding="utf-8") as f:
        f.write(f"{item},{amount}\n")
    print("Saved.")

total = 0
try:
    with open(FILE, encoding="utf-8") as f:
        for line in f:
            name, value = line.strip().rsplit(",", 1)
            total += float(value)
            print(f"{name:15} ₹{float(value):9.2f}")
    print(f"{'Total':15} ₹{total:9.2f}")
except FileNotFoundError:
    print("No expenses saved yet.")
```

Each run adds one line, such as `Bus pass,350.0`, and then lists everything saved so far with a total. `rsplit(",", 1)` splits the line once, from the right, so an item such as `pens`, `pencils` still works. In an f-string, `{name:15}` pads the name to 15 characters and `{total:9.2f}` uses 9 characters with two decimals, so the columns line up.

### REMEMBER

Catch only the errors you expect, such as `ValueError`. Never hide every error with a bare `except:`, or you will not find your own mistakes.

## Project: a class marks report

This last program uses almost everything in the book: a tuple, a dictionary, a function, a loop, decisions, f-strings and a file. It works out each student's total, percentage and grade, finds the topper, and saves a report.

### PYTHON

```
# Program 15: Marks report
def grade(percent):
    if percent >= 90:
        return "A"
    elif percent >= 75:
        return "B"
    elif percent >= 60:
        return "C"
    elif percent >= 40:
        return "D"
    else:
        return "E"

subjects = ("Hindi", "English", "Maths", "Science", "Social Science")
students = {
    "Aarav": (78, 85, 92, 88, 74),
    "Diya": (91, 89, 95, 97, 90),
    "Imran": (55, 62, 48, 60, 58),
    "Simran": (68, 74, 81, 70, 77),
}
max_total = len(subjects) * 100

lines = [f"{'Name':10}{'Total':>7}{'Percent':>9} Grade"]
topper = ""
best = -1
for name, marks in students.items():
    total = sum(marks)
    percent = total / max_total * 100
    lines.append(f"{name:10}{total:7}{percent:9.1f} {grade(percent)}")
    if total > best:
        best = total
        topper = name
lines.append(f"Topper: {topper} with {best} marks out of {max_total}")

for line in lines:
    print(line)

with open("report.txt", "w", encoding="utf-8") as f:
    f.write("\n".join(lines) + "\n")
print("Report saved to report.txt")
```

## What it prints

```
Name      Total  Percent  Grade
Aarav      417     83.4    B
Diya       462     92.4    A
Imran      283     56.6    D
Simran     370     74.0    C
Topper: Diya with 462 marks out of 500
Report saved to report.txt
```

## How it works, step by step

1. `grade()` turns a percentage into a grade, using the same example bands as Program 5.
2. `subjects` is a tuple, because the subject list does not change. `students` is a dictionary: each name is a key, and the value is a tuple of five marks.
3. `max_total` is  $5 \times 100 = 500$ , worked out from the tuple, so adding a sixth subject needs no other change.
4. The loop takes each student, adds up the marks with `sum()`, and works out the percentage.
5. `{ 'Name' : 10 }` pads text to 10 characters and `{ 'Total' : >7 }` pushes text to the right of 7 characters, so the columns line up.
6. The program keeps the highest total seen so far in `best`, along with the topper's name.
7. `"\n".join(lines)` joins all the lines with a new line between each, and the file is written in one go.

### TRY THIS

Add a fifth student. Then add a line that prints the class average percentage. Finally, change the program to ask for each student's name and marks with `input()` instead of typing them into the dictionary.

## Where to go next

You now know the building blocks that every programming language shares. Practise a little every day. Change the programs in this book, break them on purpose, and read the error messages. The official tutorial at [python.org](https://python.org) is a good next step, and so is building small programs of your own, such as a quiz, a bill maker or a timetable printer.

### CHECK YOURSELF

## Practice questions

1. What is IDLE, and what are its two kinds of window used for?

2. Why does `print("5" + "3")` show 53 and not 8?
3. What type does `input()` always give back, and how do you turn the answer into a whole number?
4. Which of these variable names are not allowed: `total2`, `2total`, `total marks`, `total_marks`?
5. What are the results of `17 // 5`, `17 % 5` and `17 / 5`?
6. Work out `3 + 4 * 2 ** 2` the way Python does.
7. If `city = "Hyderabad"`, what are `city[0]`, `city[-1]` and `city[0:5]`?
8. Write one line that prints a name stored in `name` in capital letters.
9. What is the difference between `=` and `==`?
10. Write an `if` statement that prints "Adult" when `age` is 18 or more, and "Minor" otherwise.
11. Why is indentation important in Python?
12. What numbers does `range(3, 15, 3)` give?
13. What is an infinite loop, and how do you stop one?
14. Write a `for` loop that prints the squares of the numbers 1 to 5.
15. What does `break` do inside a loop?
16. Given `nums = [4, 9, 2, 7]`, what do `max(nums)`, `sum(nums)` and `nums[2]` give?
17. How is a tuple different from a list?
18. Make a dictionary of three states and their capitals, then print the capital of one of them.
19. Write a function `is_even(n)` that returns `True` when `n` is even and `False` otherwise.
20. What is the difference between a parameter and an argument?
21. Which module and function would you use to pick a random whole number from 1 to 100?
22. What happens to a file's old contents when you open it with mode `"w"`, and with mode `"a"`?
23. Write a `try` and `except` block that asks for a number and prints "Please type digits only" if the user types letters.
24. In Program 9, why does the loop stop when `d * d` is greater than `n`?

## ANSWER KEY

# Answers

---

1. IDLE is the simple editor that comes with Python. The Shell window (with `>>>`) runs one line at a time; the editor window is for writing and saving whole programs.
2. Both values are in quotes, so they are strings. `+` joins strings, so "5" and "3" become "53".
3. It always gives back a string. Use `int()`, for example `n = int(input("Number: "))`.

4. `2total` (starts with a digit) and `total` marks (has a space) are not allowed. The other two are fine.
5. `17 // 5` is 3, `17 % 5` is 2 and `17 / 5` is 3.4.
6. Power first: `2 ** 2` is 4. Then multiply: `4 * 4` is 16. Then add: `3 + 16` is 19.
7. `city[0]` is "H", `city[-1]` is "d" and `city[0:5]` is "Hyder".
8. `print(name.upper())`
9. `=` stores a value in a variable. `==` compares two values and gives True or False.
10. `if age >= 18:` then indented `print("Adult")`, then `else:` with indented `print("Minor")`.
11. Python uses indentation to know which lines belong to an `if`, a loop or a function. Wrong indentation causes an error or makes the wrong lines run.
12. 3, 6, 9 and 12. The stop value 15 is not included.
13. A loop whose condition never becomes false, so it runs for ever. Press Ctrl+C to stop it, then fix the code so the condition can change.
14. `for i in range(1, 6):` then indented `print(i ** 2)`. It prints 1, 4, 9, 16 and 25.
15. It leaves the loop at once, even if the loop's condition is still true.
16. `max(nums)` is 9, `sum(nums)` is 22 and `nums[2]` is 2.
17. A list can be changed after it is made; a tuple cannot. Lists use square brackets and tuples use round brackets.
18. For example `capitals = {"Kerala": "Thiruvananthapuram", "Bihar": "Patna", "Assam": "Dispur"}` and then `print(capitals["Bihar"])`, which prints Patna.
19. `def is_even(n):` then indented `return n % 2 == 0`. The comparison already gives True or False.
20. A parameter is the variable name in the function's definition. An argument is the actual value passed when the function is called.
21. The `random` module: `random.randint(1, 100)`, after `import random`.
22. Mode "w" wipes the old contents and starts empty. Mode "a" keeps them and adds new text at the end.
23. `try:` then indented `n = int(input("Number: "))`, then `except ValueError:` with indented `print("Please type digits only")`.
24. If `n` had a factor larger than its square root, it would be paired with a factor smaller than the square root, which the loop would already have found. So checking further is wasted work.

## WORD LIST

# Glossary

## Argument

the actual value you pass to a function when you call it.

## Comment

a note in the code that starts with `#`, which Python ignores.

**Condition**

a question in the code whose answer is True or False.

---

**Dictionary**

a collection of key and value pairs inside curly brackets.

---

**Error**

a problem that stops a program and prints a message.

---

**Float**

a number with a decimal point, such as 3.5.

---

**Function**

a named block of code that does one job and can be used many times.

---

**IDLE**

the simple editor and Shell that come with Python.

---

**Indentation**

spaces at the start of a line that show which block it belongs to.

---

**Index**

the position number of an item in a string or list, starting from 0.

---

**Input**

information the user gives to a program, often typed at the keyboard.

---

**Integer**

a whole number with no decimal part, called `int` in Python.

---

**List**

an ordered collection of items inside square brackets that can be changed.

---

**Loop**

code that repeats, made with `while` or `for`.

---

**Method**

a function that belongs to a value and is called with a dot, such as `.upper()`.

---

**Module**

a file of ready-made functions you can bring in with `import`.

---

**Parameter**

a variable in a function's definition that receives an argument.

---

**Program**

a list of instructions that a computer follows in order.

---

**Slice**

a part of a string or list taken with square brackets and a colon, such as `word[1:4]`.

---

**String**

text inside quotes, such as "Chennai".

---

**Traceback**

the error report Python prints, whose last line names the error.

---

**Tuple**

an ordered collection inside round brackets that cannot be changed.

---

**Variable**

a name that stores a value, like a labelled box.

---



# Gurukul Technicals Institute

Computer education institute in Delhi, with authorised member centres, robotics classes and science projects, working models and DIY kits.

## COURSES

Computer & Office Skills · Accounting & Business Software · Programming & Web Development · Design & Digital Skills · Hardware & Networking · Robotics & Coding · Other Programmes

## PROJECTS AND KITS

Ready-made school projects · DIY and STEM kits · Robotics and coding · Components and spares · Engineering projects

## CONTACT

**Head office:** A-10, Chief Commissioner Colony, opposite Rana Pratap Bagh, Delhi 110007

**Phone:** 011 2721 1512 · **Mobile and WhatsApp:** +91 97116 99511

**Email:** [engineergurukul@yahoo.co.in](mailto:engineergurukul@yahoo.co.in)

**Website:** [engineergurukul.com](http://engineergurukul.com) · **Instagram:** [@science\\_projects\\_club](https://www.instagram.com/science_projects_club)

---

**More free ebooks** for computer courses, robotics and science projects: [engineergurukul.com/resources/ebooks/](http://engineergurukul.com/resources/ebooks/)

Free to read, print and share. Not for sale.