BEGINNER FRIENDLY

Python Fundamentals

From Zero to Programmer

Learn the world's most popular programming language

Module 1: Getting Started with Python

Module 2: Data Types & Variables

Module 3: Control Flow & Functions

Module 4: Collections & Loops

Why Learn Python?

Easy to Learn

Clean, readable syntax that reads like English. Perfect for beginners!

Most Popular

#1 programming language. Used by Google, Netflix, Instagram, and NASA.

High Demand

Top-paying skill in tech. Average salary $120k+ in the US.

Python is used for: Web Development, Data Science, AI/ML, Automation, Game Development, and more!

Your First Python Program

The classic "Hello, World!" - every programmer's first step:

hello.py

```
print("Hello, World!")
```

That's it! One line of code. In other languages, this could be 5-10 lines.

> **Try it yourself:** Open Python and type this command. Watch the magic happen!

Variables: Storing Information

Variables are like labeled boxes that store data:

Creating Variables

```
# Storing different types of data
name = "Alice"
age = 25
height = 5.6
is_student = True

# Using variables
print(f"Hi, I'm {name}!")
print(f"I'm {age} years old")
```

Key Points:

- No need to declare types
- Use descriptive names
- snake_case is preferred
- Case sensitive (Name != name)

Naming tip: Choose names that describe what the variable holds!

Python Data Types

Strings (str)

```
"Hello World"
'Python is fun'
"""Multi-line
string"""
```

Text data in quotes

Numbers

```
# Integer
age = 25

# Float (decimal)
price = 19.99
```

Whole & decimal numbers

Boolean (bool)

```
is_active = True
is_admin = False

# Comparisons
5 > 3  # True
```

True or False values

Operators: Doing Math & More

Arithmetic Operators

```
a = 10
b = 3

print(a + b)   # 13 (addition)
print(a - b)   # 7  (subtraction)
print(a * b)   # 30 (multiplication)
print(a / b)   # 3.33 (division)
print(a // b)  # 3  (floor division)
print(a % b)   # 1  (modulo/remainder)
print(a ** b)  # 1000 (power)
```

Comparison Operators

```
x = 5
y = 10

print(x == y)  # False (equal?)
print(x != y)  # True  (not equal?)
print(x < y)   # True  (less than?)
print(x > y)   # False (greater?)
print(x <= y)  # True  (less or equal?)
print(x >= y)  # False (greater or equal?)
```

Working with Strings

String Operations

```
name = "Python"

# Length
len(name)  # 6

# Indexing (starts at 0!)
name[0]    # 'P'
name[-1]   # 'n' (last char)

# Slicing
name[0:3]  # 'Pyt'
name[2:]   # 'thon'

# Methods
name.upper()   # 'PYTHON'
name.lower()   # 'python'
name.replace('P', 'J')  # 'Jython'
```

String Formatting (f-strings)

```
name = "Alice"
age = 30

# Modern f-string (recommended!)
msg = f"Hi {name}, you are {age}"

# Math in f-strings
print(f"Next year: {age + 1}")

# Formatting numbers
price = 49.99
print(f"Price: ${price:.2f}")
```

f-strings are the modern way to format strings. The 'f' stands for "formatted"!

Lists: Ordered Collections

Lists store multiple items in a single variable:

Creating & Using Lists

```
# Creating lists
fruits = ["apple", "banana", "cherry"]
numbers = [1, 2, 3, 4, 5]
mixed = [1, "hello", True, 3.14]

# Accessing items
print(fruits[0])   # 'apple'
print(fruits[-1])  # 'cherry'

# Modifying
fruits[1] = "blueberry"
fruits.append("orange")
fruits.remove("apple")

# List length
len(fruits)  # 3
```

Common List Methods

Method	Description
`append(x)`	Add item to end
`insert(i, x)`	Insert at position
`remove(x)`	Remove first match
`pop()`	Remove last item
`sort()`	Sort the list
`reverse()`	Reverse order

Dictionaries: Key-Value Pairs

Dictionaries store data as key-value pairs (like a real dictionary!):

Working with Dictionaries

```
# Creating a dictionary
person = {
    "name": "Alice",
    "age": 30,
    "city": "New York",
    "is_student": False
}

# Accessing values
print(person["name"])      # 'Alice'
print(person.get("age"))   # 30

# Adding/updating
person["email"] = "alice@example.com"
person["age"] = 31

# Removing
del person["is_student"]
```

When to Use Dictionaries?

- Storing related information
- Looking up values by name
- Representing real-world objects
- Configuration settings

Pro tip: Use `.get(key)` to avoid errors if key doesn't exist!

Conditionals: Making Decisions

Use if/elif/else to control program flow:

If Statements

```
age = 18

if age < 13:
    print("Child")
elif age < 20:
    print("Teenager")
else:
    print("Adult")

# Multiple conditions
temperature = 75
is_sunny = True

if temperature > 70 and is_sunny:
    print("Perfect beach day!")

if temperature < 32 or temperature > 100:
    print("Extreme weather!")
```

Logical Operators

Operator	Meaning
`and`	Both must be True
`or`	At least one True
`not`	Inverts the value

Indentation matters! Python uses indentation (4 spaces) to define code blocks.

Loops: Repeating Actions

For Loops

```
# Loop through a list
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(f"I love {fruit}!")

# Loop with range
for i in range(5):
    print(i)  # 0, 1, 2, 3, 4

# Loop with index
for i, fruit in enumerate(fruits):
    print(f"{i}: {fruit}")
```

While Loops

```
# While loop
count = 0
while count < 5:
    print(count)
    count += 1

# Break and Continue
for i in range(10):
    if i == 3:
        continue  # skip 3
    if i == 7:
        break     # stop at 7
    print(i)
```

> Use **for** when you know the iterations. Use **while** when you don't!

Functions: Reusable Code

Functions let you write code once and use it many times:

Defining Functions

```
# Simple function
def greet(name):
    return f"Hello, {name}!"

print(greet("Alice"))  # Hello, Alice!

# Default parameters
def greet(name, greeting="Hello"):
    return f"{greeting}, {name}!"

print(greet("Bob"))           # Hello, Bob!
print(greet("Bob", "Hi"))     # Hi, Bob!

# Multiple return values
def get_stats(numbers):
    return min(numbers), max(numbers)

low, high = get_stats([1, 5, 3, 9, 2])
```

Why Use Functions?

- **DRY:** Don't Repeat Yourself
- **Organize:** Break code into logical pieces
- **Reuse:** Write once, use many times
- **Test:** Easier to test small functions

Good function names describe what the function does: `calculate_total()`, `send_email()`, `get_user()`

Error Handling: Try/Except

Gracefully handle errors instead of crashing:

Try/Except Blocks

```
# Basic error handling
try:
    number = int(input("Enter a number: "))
    result = 10 / number
    print(f"Result: {result}")
except ValueError:
    print("That's not a valid number!")
except ZeroDivisionError:
    print("Can't divide by zero!")
except Exception as e:
    print(f"Something went wrong: {e}")
finally:
    print("This always runs!")
```

Common Exceptions

Exception	When it happens
`ValueError`	Wrong type of value
`TypeError`	Wrong type operation
`KeyError`	Dict key not found
`IndexError`	List index out of range
`FileNotFoundError`	File doesn't exist

Working with Files

Reading Files

```
# Reading entire file
with open("data.txt", "r") as file:
    content = file.read()
    print(content)

# Reading line by line
with open("data.txt", "r") as file:
    for line in file:
        print(line.strip())
```

Writing Files

```
# Writing to a file
with open("output.txt", "w") as file:
    file.write("Hello, File!\n")
    file.write("Second line")

# Appending to a file
with open("log.txt", "a") as file:
    file.write("New log entry\n")
```

Always use `with` statement! It automatically closes the file when done, even if an error occurs.

Modules: Extending Python

Import pre-built functionality to supercharge your code:

Using Modules

```
# Import entire module
import math
print(math.sqrt(16))  # 4.0
print(math.pi)        # 3.14159...

# Import specific functions
from random import randint, choice
print(randint(1, 100))
print(choice(["a", "b", "c"]))

# Import with alias
import datetime as dt
today = dt.date.today()
print(today)
```

Popular Built-in Modules

Module	Purpose
`math`	Mathematical functions
`random`	Random numbers
`datetime`	Date & time
`os`	Operating system
`json`	JSON parsing
`re`	Regular expressions

List Comprehensions: Python Magic

A concise way to create lists:

Traditional Way

```
# Create list of squares
squares = []
for x in range(10):
    squares.append(x ** 2)

# Filter even numbers
evens = []
for x in range(20):
    if x % 2 == 0:
        evens.append(x)
```

With List Comprehension

```
# Create list of squares
squares = [x ** 2 for x in range(10)]

# Filter even numbers
evens = [x for x in range(20) if x % 2 == 0]

# Transform data
names = ["alice", "bob", "charlie"]
upper = [name.upper() for name in names]
# ['ALICE', 'BOB', 'CHARLIE']
```

Pattern: `[expression for item in iterable if condition]`

What You've Learned

Basics

- Variables & data types
- Operators
- String manipulation
- Print & input

Data Structures

- Lists
- Dictionaries
- List comprehensions
- Slicing

Control Flow

- If/elif/else
- For & while loops
- Functions
- Error handling

Next Steps:

- Complete the hands-on labs
- Build small projects (calculator, to-do app, quiz game)
- Explore: Web dev (Flask/Django), Data Science (Pandas), or Automation