Introduction to Python Programming

Chapter 2: Python Basics

Introduction

Now that you understand what programming is and have your environment set up, it's time to start writing actual Python code! This chapter covers the fundamental building blocks you'll use in every Python program you write.

---

1. White Space Matters in Python

This is crucial: Python is a **white space dependent** language. This means that spaces and tabs at the beginning of lines actually mean something to Python.

What Does This Mean?

In many other programming languages, indentation (spaces/tabs) is just for making code look nice. In Python, indentation is **part of the syntax** - it's how Python understands which code belongs together.

Example of how indentation matters:

```
# Correct - no indentation at the start
name = "Alice"
age = 20

# WRONG - random space at the beginning
 name = "Bob"  # This will cause an error!
```

Code Blocks:

When you indent code, you're creating a **code block** - a group of related statements that belong together. You'll see this more when we work with if statements and loops, but for now, remember:

- **Don't randomly add spaces or tabs at the beginning of lines**
- Python uses indentation to group code together
- Be consistent - use either spaces OR tabs (VS Code handles this for you)

We'll see much more practical examples of this in later chapters. For now, just be aware that **white space counts!**

---

2. Variables and Naming Conventions

Variables are containers that store data. Think of them as labeled boxes where you can put information.

Creating Variables

```
age = 25
name = "Sarah"
price = 19.99
is_student = True
```

Variable Naming Rules in Python

Python variables follow these rules:

1. **Use lowercase letters** for regular variables
2. **Use snake_case** for multi-word variables (words separated by underscores)
3. **Use UPPERCASE** for constants (values that never change)

Examples of good variable names:

```
# Good variable names (snake_case)
student_name = "John"
total_price = 99.99
is_logged_in = True
user_age = 30

# Constants (UPPERCASE)
MAX_STUDENTS = 30
PI = 3.14159
TAX_RATE = 0.05
```

Examples of poor variable names:

```
# Poor variable names
x = "John"           # Too vague - what is x?
n = "John"           # Not descriptive
a = 25               # What does 'a' represent?
studentName = "Bob"  # Wrong case (this is camelCase)
StudentName = "Sue"  # Wrong case (this is PascalCase)
```

Other Naming Conventions (For Contrast)

Python uses **snake_case**, but you'll see other conventions in other languages:

- **camelCase**: `studentName`, `totalPrice` (used in JavaScript, Java)
- **PascalCase**: `StudentName`, `TotalPrice` (used for classes in Python)

- **snake_case**: `student_name`, `total_price` (used for variables in Python)

Remember: For this course and Python in general, use **snake_case** for variables!

---

3. Comments in Python

Comments are lines in your code that Python ignores. They're notes for humans (including your future self!) to explain what the code does.

Single-Line Comments

Use the `#` symbol to create a single-line comment. Everything after `#` on that line is ignored by Python.

```
# This is a comment
age = 25  # This is also a comment

# Calculate the area of a rectangle
width = 10
height = 5
area = width * height  # area = 50
```

When to use single-line comments:

- Explain what a section of code does
- Add notes about why you made certain choices
- Temporarily disable a line of code while testing

Multi-Line Comments

Python doesn't have a specific multi-line comment syntax, but you can use multiple `#` symbols:

```
# This is a longer comment
# that spans multiple lines
# to explain something complex

age = 25
```

Docstrings (Documentation Strings)

For longer documentation, Python has a special type of string called a **docstring**. These use triple quotes (`"""` or `'''`) and are typically used to document functions, classes, and modules.

```
"""
This is a docstring.
It can span multiple lines.
It's often used at the top of files or to document functions.
"""

def calculate_area(width, height):
    """
    Calculate the area of a rectangle.

    This function takes width and height and returns the area.
    """
    return width * height
```

Important notes about docstrings:

- They're actually stored as part of the code (unlike comments)
- There are standard formats (like Google style, NumPy style) for writing them
- They're used to generate automatic documentation
- We'll learn more about docstrings when we cover functions

For now, just know that docstrings exist and are the professional way to document your code.

Comment Best Practices

```
# GOOD: Explains WHY
# Use floor division because we can't have partial students
students_per_group = total_students // num_groups

# BAD: Just repeats what the code says
# Divide total_students by num_groups
students_per_group = total_students // num_groups

# GOOD: Clarifies complex logic
# Check if user is eligible: must be 18+ AND have valid ID
if age >= 18 and has_valid_id:
    print("Access granted")
```

Remember: Comments should explain *why* you're doing something, not just *what* the code does. Good code should be readable enough that the "what" is obvious.

---

3. Comments in Python

Comments are lines in your code that Python ignores. They're notes for humans (including your future self!) to explain what the code does.

Single-Line Comments

Use the `#` symbol to create a single-line comment. Everything after `#` on that line is ignored by Python.

```
# This is a comment
age = 25  # This is also a comment

# Calculate the area of a rectangle
width = 10
height = 5
area = width * height  # area = 50
```

When to use single-line comments:

- Explain what a section of code does
- Add notes about why you made certain choices
- Temporarily disable a line of code while testing

Multi-Line Comments

Python doesn't have a specific multi-line comment syntax, but you can use multiple `#` symbols:

```
# This is a longer comment
# that spans multiple lines
# to explain something complex

age = 25
```

Docstrings (Documentation Strings)

For longer documentation, Python has a special type of string called a **docstring**. These use triple quotes (`"""` or `'''`) and are typically used to document functions, classes, and modules.

```
"""
This is a docstring.
It can span multiple lines.
It's often used at the top of files or to document functions.
"""

def calculate_area(width, height):
    """
    Calculate the area of a rectangle.

    This function takes width and height and returns the area.
    """
    return width * height
```

Important notes about docstrings:

- They're actually stored as part of the code (unlike comments)
- There are standard formats (like Google style, NumPy style) for writing them
- They're used to generate automatic documentation
- We'll learn more about docstrings when we cover functions

For now, just know that docstrings exist and are the professional way to document your code.

Comment Best Practices

```
# GOOD: Explains WHY
# Use floor division because we can't have partial students
students_per_group = total_students // num_groups

# BAD: Just repeats what the code says
# Divide total_students by num_groups
students_per_group = total_students // num_groups

# GOOD: Clarifies complex logic
# Check if user is eligible: must be 18+ AND have valid ID
if age >= 18 and has_valid_id:
    print("Access granted")
```

Remember: Comments should explain *why* you're doing something, not just *what* the code does. Good code should be readable enough that the "what" is obvious.

---

4. Data Types

Python has special words called **keywords** that have specific meanings in the language. You **cannot** use these as variable names.

What Are Keywords?

Keywords are words that Python uses for its own purposes. Examples include:

```
for    if      while   def     class   return
in     and     or      not     True    False
import from    as      try     except  finally
```

How to Recognize Keywords

Most code editors (like VS Code) will **color code** keywords differently from regular text.

Try this experiment in VS Code:

1. Type the word `for` - notice it has a special color (usually purple or blue)
2. Type the word `apple` - it will be a different color (usually white or black)
3. Any word that has the same color as `for` is a keyword!

What happens if you try to use a keyword as a variable?

```
for = 5  # ERROR! 'for' is a keyword
if = 10  # ERROR! 'if' is a keyword
```

You'll get a syntax error because Python thinks you're trying to use these words for their special purposes.

Don't worry about memorizing all keywords! Your code editor will let you know when you accidentally use one by coloring it differently.

---

4. Data Types

Python variables can hold different **types** of data. The main basic types you need to know are:

Integer (int)

Whole numbers, positive or negative, with no decimal point.

```
age = 25
temperature = -10
year = 2024
count = 0
```

Float (float)

Numbers with decimal points.

```
price = 19.99
temperature = 98.6
pi = 3.14159
grade = 87.5
```

String (str)

Text, surrounded by quotes (single or double).

```
name = "Alice"
message = 'Hello, World!'
address = "123 Main Street"
```

Boolean (bool)

True or False values (note the capitalization!).

```
is_student = True
is_raining = False
has_license = True
```

Dynamic Typing

Important: Python variables can change type! The same variable can hold different types at different times:

```
x = 5           # x is an integer
print(x)        # 5

x = "hello"     # now x is a string
print(x)        # hello

x = 3.14        # now x is a float
print(x)        # 3.14
```

This is called **dynamic typing** - the type can change as the program runs. This is flexible but requires you to be careful!

Checking the Type

You can check what type a variable is using the `type()` function:

```
age = 25
print(type(age))          # <class 'int'>

price = 19.99
print(type(price))        # <class 'float'>

name = "Bob"
print(type(name))         # <class 'str'>

is_student = True
print(type(is_student))   # <class 'bool'>
```

---

5. Basic Math Operations

Python can perform mathematical calculations using standard operators.

Basic Operators

```
# Addition
result = 5 + 3        # 8

# Subtraction
result = 10 - 4       # 6

# Multiplication
result = 6 * 7        # 42

# Division (always gives a float)
result = 15 / 3       # 5.0
result = 10 / 4       # 2.5

# Floor Division (rounds down to nearest integer)
result = 10 // 3      # 3
result = 15 // 4      # 3

# Modulus (remainder after division)
result = 10 % 3       # 1
result = 17 % 5       # 2

# Exponent (power)
result = 2 ** 3       # 8 (2 to the power of 3)
result = 5 ** 2       # 25 (5 squared)
```

Using Variables in Calculations

```
price = 100
tax_rate = 0.07
tax = price * tax_rate       # 7.0
total = price + tax          # 107.0

width = 10
height = 5
area = width * height        # 50

# You can also update variables
count = 5
count = count + 1            # 6

# Shorthand for updating
count += 1    # Same as: count = count + 1
count -= 2    # Same as: count = count - 2
count *= 3    # Same as: count = count * 3
count /= 2    # Same as: count = count / 2
```

Order of Operations

Python follows the standard mathematical order of operations (PEMDAS):

1. **P**arentheses
2. **E**xponents
3. **M**ultiplication and **D**ivision (left to right)
4. **A**ddition and **S**ubtraction (left to right)

```
result = 5 + 3 * 2          # 11 (multiplication first)
result = (5 + 3) * 2        # 16 (parentheses first)
result = 10 / 2 + 3         # 8.0 (division first)
result = 2 ** 3 + 5         # 13 (exponent first)
```

---

6. Type Casting (Converting Between Types)

Sometimes you need to convert a value from one type to another. This is called **type casting** or **type conversion**.

Casting Functions

```
# String to Integer
age_str = "25"
age_int = int(age_str)      # 25 (as integer)
```

```
# String to Float
price_str = "19.99"
price_float = float(price_str)  # 19.99 (as float)

# Integer to String
age = 25
age_str = str(age)          # "25" (as string)

# Float to Integer (loses decimal part!)
price = 19.99
price_int = int(price)      # 19 (decimal part removed)

# Integer to Float
age = 25
age_float = float(age)      # 25.0
```

Lossy Conversions

Be careful! Some conversions lose data - this is called a **lossy conversion**:

```
# Float to Integer - LOSES decimal places
price = 19.99
price_int = int(price)     # 19 (lost the .99!)

score = 87.8
score_int = int(score)     # 87 (lost the .8!)

# This is NOT rounding - it just cuts off the decimal
value = 9.9
value_int = int(value)     # 9 (not 10!)
```

Implicit (Automatic) Casting

Sometimes Python automatically converts types for you:

```
# Integer + Float = Float
result = 5 + 2.5           # 7.5 (Python converts 5 to 5.0)

# Division always returns Float
result = 10 / 2            # 5.0 (not 5!)

# Mixing types in calculation
x = 10        # integer
y = 3.5       # float
z = x * y     # 35.0 (Python converts x to float)
```

When Do You Need to Cast?

You'll often need to cast when:

1. **Getting user input** (input always gives you a string)

```
age = input("Enter age: ")      # age is a string "25"
age = int(age)                  # now age is integer 25
```

2. **Combining different types**

```
age = 25
message = "You are " + str(age) + " years old"
# Must convert age to string to concatenate
```

3. **Performing calculations with input**

```
num1 = input("Enter first number: ")
num2 = input("Enter second number: ")
total = int(num1) + int(num2)  # Must convert to do math
```

Type Casting Examples

```
# Example 1: Temperature conversion
celsius_str = "25"
celsius = float(celsius_str)
fahrenheit = (celsius * 9/5) + 32
print(fahrenheit)  # 77.0

# Example 2: Calculating with string numbers
price1 = "10.99"
price2 = "5.50"
total = float(price1) + float(price2)  # 16.49

# Example 3: Integer division vs regular division
apples = 17
people = 5
per_person = apples // people    # 3 (floor division)
remainder = apples % people      # 2 (modulus)
```

---

7. Putting It All Together: Examples

Example 1: Basic Calculations

```python
# Calculate area of a rectangle
width = 10
height = 5
area = width * height
print(area)  # 50

# Calculate with floats
price = 19.99
quantity = 3
total = price * quantity
print(total)  # 59.97
```

Example 2: Working with Different Types

```python
# Mixing types
name = "Alice"
age = 25
is_student = True

# These are all different types!
print(type(name))       # <class 'str'>
print(type(age))        # <class 'int'>
print(type(is_student)) # <class 'bool'>
```

Example 3: Type Conversion in Action

```python
# User input is always string
age_input = "25"
years_until_30 = 30 - int(age_input)
print(years_until_30)  # 5

# Converting for different operations
score = 87.5
score_int = int(score)              # 87
score_str = str(score)              # "87.5"
print(type(score_int))              # <class 'int'>
print(type(score_str))              # <class 'str'>
```

Key Takeaways

✓ **White space matters** in Python - don't add random spaces at the start of lines

✓ **Variables use snake_case**, constants use UPPERCASE

✓ **Comments explain your code** - use # for single-line, """ for docstrings

✓ **Keywords are reserved words** - your editor will color them differently

✓ **Four main data types:** int, float, str, bool

✓ **Python is dynamically typed** - variables can change type

✓ **Math operators:** +, -, *, /, //, %, **

✓ **Type casting:** int(), float(), str() - watch for lossy conversions!

✓ **Some conversions lose data** - like float to int drops decimals

---

Reflection Questions

1. Why is white space important in Python?
2. What naming convention does Python use for variables?
3. Can you name the four basic data types we covered?
4. What's the difference between `/` and `//`?
5. What happens when you convert a float to an int?
6. How can you check what type a variable is?

Practice Exercises

Try these on your own to reinforce what you've learned:

1. Create variables for your name, age, and height. Use appropriate types.
2. Calculate the area of a circle with radius 5 (use 3.14159 for pi).
3. Convert a string "100" to an integer and multiply it by 5.
4. What is 17 divided by 5? What is 17 floor-divided by 5? What is 17 modulo 5?
5. Create a variable with value 99.9, convert it to int. What do you get?