The Python Programming Language

Why learn Python?

Popular – used by millions of developers.

Well-supported – large libraries of support code.

Useful – used in every major industry.

Rephactor

The Python Programming Language

Python is a **general-purpose** language.

It embraces multiple development paradigms:

- **Procedural programming**
- **Object-oriented programming**
- **Functional programming**

Its versatility is one of its strengths.

Hello, World!

Python is an **interpreted** language.

That means Python code is run one statement at a time using a software tool called the Python **interpreter**.

Other languages, such as Java and C++, are **compiled** languages.

Programs in those languages must be first translated into an executable form by a **compiler** before they can be run.

There is no preprocessing step with a Python program. You can run it immediately.

Downside is Python is slower than a compiled language.

Comments

A program should contain **comments** to explain the program's purpose and processing.

Comments are intended for the human reader – they have no effect on a program.

A Python comment begins with a **#** and continues until the end of the line.

```
# This is a comment
```

It might be put on the end of a line of code:

```
balance = balance - fees  # deduct monthly fees
```

Comments

```
#
# Function to print a quote from The Simpsons episode titled
# Bart vs. Thanksgiving.
#
def print_simpsons_quote():
    print("Operator, what's the number for 9-1-1?")  # s2, ep7
    print('    - Homer Simpson')

# Main program
print_simpsons_quote()
```

```
Operator, what's the number for 9-1-1?
    - Homer Simpson
```

A **block comment** is a series of comments that work together.

Comments

A comment can also be added to code using a **docstring**.

A docstring is simply an isolated string that contains a comment.

```
'This is a docstring'
```

Like all Python strings, it can be surrounded by single or double quote characters.

Since the string is not being used in any way, it has no effect on the program. It is ignored by the Python interpreter.

Comments

Docstrings are often used to document the purpose of a function:

```
def compute_perimiter(side_length):
    'Computes the perimeter of the square.'
    perimeter = side_length * 4
    return perimeter
```

Longer docstrings that span more than one line use **triple quotes** at either end:

```
'''
  This is a multi-line docstring, used as a block
  comment for larger explanations.
'''
```

Variables

A variable name, like any other name you make up in a program, is an **identifier**.

Identifier names can be composed of letters (A-Z and a-z), digits (0-9), and the underscore character (_).

An identifier cannot begin with a digit.

Valid	Invalid
`sum`	`1st_place`
`last_name`	`total$`
`label4`	`#yolo`

Rephactor

Variables

A **convention** is a guideline that programmers agree to follow to make their code easier to read.

The Python interpreter won't care if you follow conventions, but other programmers certainly will.

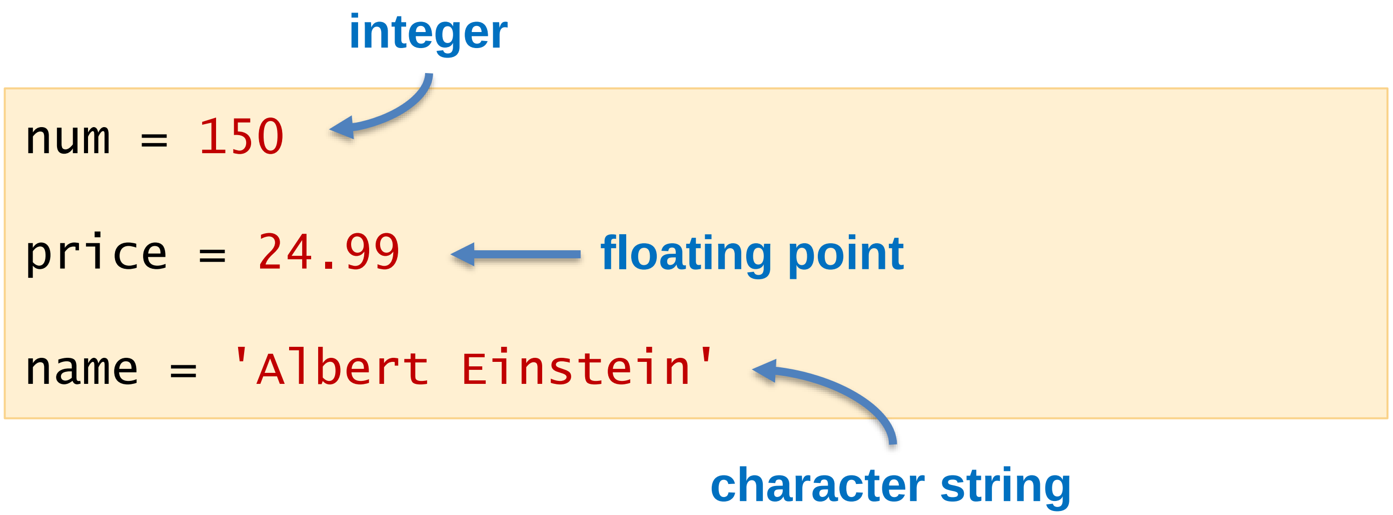
By **convention**, variables in Python are in lowercase, with words separated by underscores.

Variables should also be concise but descriptive.

Valid and Good	Valid but NOT Good
`highest`	`Highest`
`inventory_limit`	`inventoryLimit`
`current_stock_value`	`cur_st_val`

Variables

Each value has a **data type** that determines the kind of operations that can be performed on it.



```
num = 150

price = 24.99

name = 'Albert Einstein'
```

The data type is associated with the value, not the variable.

In Python, any variable can be assigned a value of any type.

Constants

A **constant** is like a variable, except that its value never changes.

By convention, a Python constant name uses all uppercase letters.

```
PINTS_PER_GALLON = 8
MAX_OCCUPANCY = 650
MIN_TAX_RATE = 0.04
NAME_PLACEHOLDER = 'John Doe'
```

This naming convention distinguishes constants from regular variables, which use lowercase letters.

Constants

In Python, a constant is not really constant.

In some languages, attempting to change the value of a constant will cause an error.

A Python constant is just a regular variable that follows a different naming convention.

It is not enforced by the rules of the language.

But you should use constants to covey the idea that the value should not be changed.

Numeric Expressions

An **expression** is a combination of operators and operands.

Numeric expressions make use of the arithmetic operators:

Addition	`+`
Subtraction	`-`
Multiplication	`*`
Division	`/`
Floor Division	`//`
Modulus	`%`
Exponentiation	`**`

They all operate on integer or floating-point operands.

Numeric Expressions

The **division** operator (/) always produces a floating-point result.

`8 / 16`	equals	`0.5`
`12 / 5`	equals	`2.4`
`14 / 3`	equals	`4.666666666666667`
`15 / 3`	equals	`5.0`
`23.57 / 18.4`	equals	`1.2809782608695652`

This operation is sometimes called **true division**.

Numeric Expressions

The **floor division** operator (`//`) is the quotient after dividing one number by another. Any leftover is discarded.

If both operands are integers, the result is an integer. If either, or both, are floating-point numbers, the result is floating-point.

`14 // 3`	equals	`4`
`14.0 // 3`	equals	`4.0`
`12 // 5`	equals	`2`
`3 // 7`	equals	`0`

Floor division is sometimes called **integer division**.

Rephactor

Numeric Expressions

The **modulus** operator (%) returns the remainder after dividing the second operand into the first.

If either operand is floating-point, the result is floating-point, too.

`14 % 3`	equals	`2`
`8 % 12.0`	equals	`8.0`

```
if total % 5 == 0:
    print(total, 'is evenly divisible by 5.’)

if num % 2 == 0:
    print(num, 'is even')
```

Modulus is sometimes called the **remainder** operator.

Rephactor

Numeric Expressions

The **exponentiation** operator (`**`) returns the result of raising the first operand to the power of the second.

`5 ** 2`	equals	`25`
`2.25 ** 3`	equals	`11.390625`
`2 ** 4.1`	equals	`17.148375400580687`

You can also perform exponentiation using a built-in function named **pow** or a function in the **math** module named **pow**.

```
import math

num1 = pow(2, 3)
num2 = math.pow(3, 5)
```

The math Module

The Python mathematical operators only do basic arithmetic.

Additional math operations can be performed using some built-in functions and the functions of the **math module**.

The **math** module is part of the Python Standard Library.

It must be **imported** before it can be used.

```
import math

result = abs(num) + math.sqrt(3)
```

This calls the built-in function **abs** and the **sqrt** function of the **math** module.

The math Module

Here are the built-in functions that perform math operations.

abs(x)	Returns the absolute value of x.
min(x, y)	Returns the smaller of x and y.
max(x, y)	Returns the larger of x and y.

round(x)
round(x, digits)
	Returns x to the nearest integer or to the specified number of digits.

```
print(abs(-5))
print(min(18, 12))
print(round(7.892, 1))
```

```
5
12
7.9
```

Rephactor

The math Module

The **sqrt** function returns the square root of its argument.

The **pow** function returns the first argument raised to the power represented by the second argument.

```
print(math.sqrt(25))
print(math.pow(2, 3))
```

```
5.0
8.0
```

There is also a built-in version of the **pow** function, as well as the exponentiation operator (**).

The **pow** function of the math module always returns a floating-point result.

Augmented Assignments

An augmented assignment operator combines an operation with assignment.

It's a succinct way to update a variable based on its current value.

```
num += 1
```

is equivalent to:

```
num = num + 1
```

Rephactor

Augmented Assignments

You can always use the fully expanded assignment, but augmented assignment is sometimes more convenient.

Here's another example:

```
balance += deposit
```

The += operator works for string concatenation, too:

```
word = 'fire'
word += 'truck'
print(word)
```

```
firetruck
```

Rephactor

Augmented Assignments

There is an augmented assignment operator for all Python numeric operators.

`value -= 100`	=	`value = value - 100`
`value *= 100`	=	`value = value * 100`
`value /= 100`	=	`value = value / 100`
`value //= 100`	=	`value = value // 100`
`value %= 100`	=	`value = value % 100`

The if Statement

An **if statement** is used to make a decision.

In particular, it is used to determine which program statement to execute next.

It evaluates a boolean **condition** and only executes the **body** of the **if** statement if the condition is true.

```
if side > 0:
    perimeter = 4 * side
```

The **if** statement is sometimes referred to as a **selection** statement or a **conditional** statement.

The if Statement

The condition must be followed by a colon.

The statement(s) in the body must all be indented the same amount. By convention, the indentation should be four spaces.

A list of statements indented the same amount is sometimes called a **statement block**.

```python
if side > 0:
    perimeter = 4 * side
    print('The perimeter is', perimeter)

print('Moving on...')
```

If the condition is true, all indented statements are executed.

The if Statement

If the condition is long, you can put parentheses around it and break it over multiple lines:

```
if (attendance_count > capacity
        or group_count > MAX_GROUP_COUNT
        or temperature > 85):
    print('We need more room!')
```

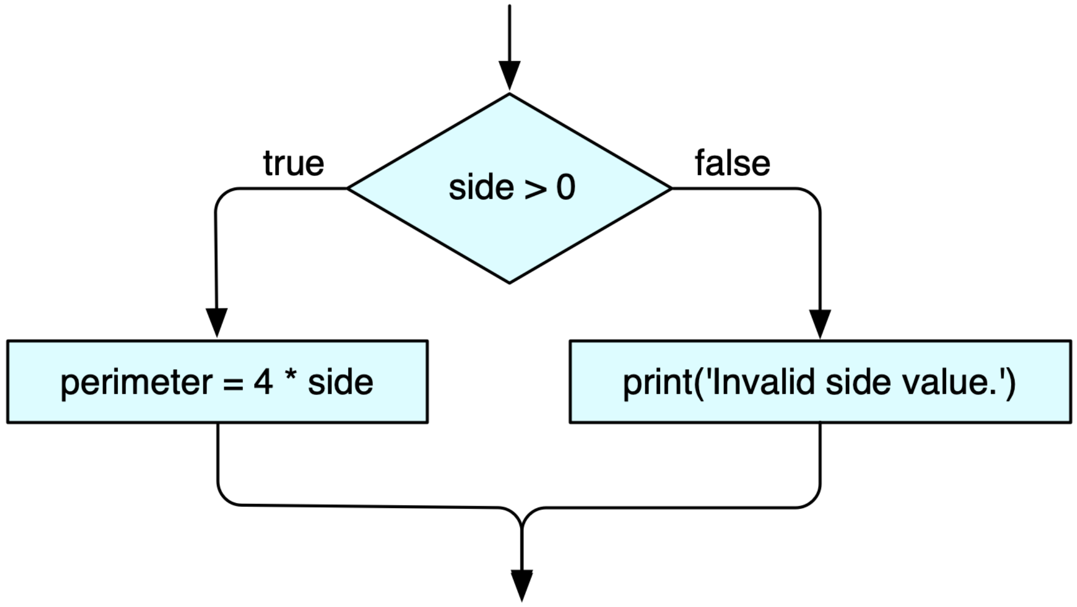
Indenting the condition lines to a different level than the body statements will improve readability.

Rephactor

The if Statement

An optional **else clause** specifies what should be done if the condition is not true.

```
if side > 0:
    perimeter = 4 * side
else:
    print('Invalid side value.')
```



Rephactor

The if Statement

The **elif clause** (short for "else if") can be used to represent a series of **if** statements.

```
if score >= 90:
    grade = 'A'
elif score >= 80:
    grade = 'B'
elif score >= 70:
    grade = 'C'
elif score >= 60:
    grade = 'D'
else:
    grade = 'F'
```

An **if** statement can have only one **else** clause, but multiple **elif** clauses.

Hello, World!

The classic first program to write when learning a new language prints a simple greeting to the world:

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

When executed, the **print statement** writes the text to the **console window**.

```
Hello, World!
```

Rephactor

Hello, World!

A **print** statement is a call to a **function**, which is a separate piece of predefined code.

The **print** function is part of a large library of code you can use in any Python program.

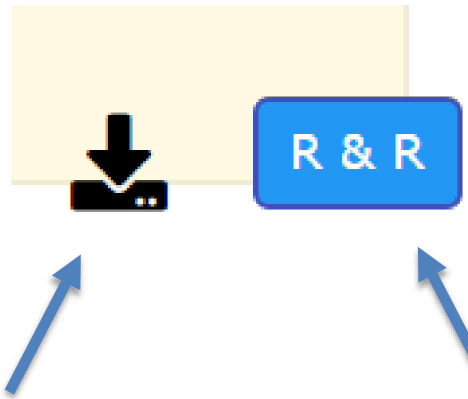
```
print('Tell me and I forget.’)
print('Show me and I remember.’)
print('Involve me and I understand.')
```

```
Tell me and I forget.
Show me and I remember.
Involve me and I understand.
```

https://docs.python.org/3/library/functions.html#print

Hello, World!

Make use of the download and R & R buttons on code in the textbook.



Download the program to your computer

Run & Revise the code right in the browser