MASTER MEIM 2022-2023

Python Programming Course
Lesson 2

Variables , Types, Strings

Lesson given by Prof. Mariacarla Staffa

Department of Science and Technology of the University of Naples Parthenope

1. Introduction

- Numbers and character strings are important data types in any Python program
  - These are the fundamental building blocks we use to build more complex data structures
- In this lesson, you will learn how to work with numbers and text.
- We will write several simple programs that use them

At the end of the lesson you will be able to

- To declare and initialize variables and constants
- To understand the properties and limitations of integers and floating-point numbers
- To appreciate the importance of comments and good code layout
- To write arithmetic expressions and assignment statements
- To create programs that read, and process inputs, and display the results
- To learn how to use Python strings

Contents

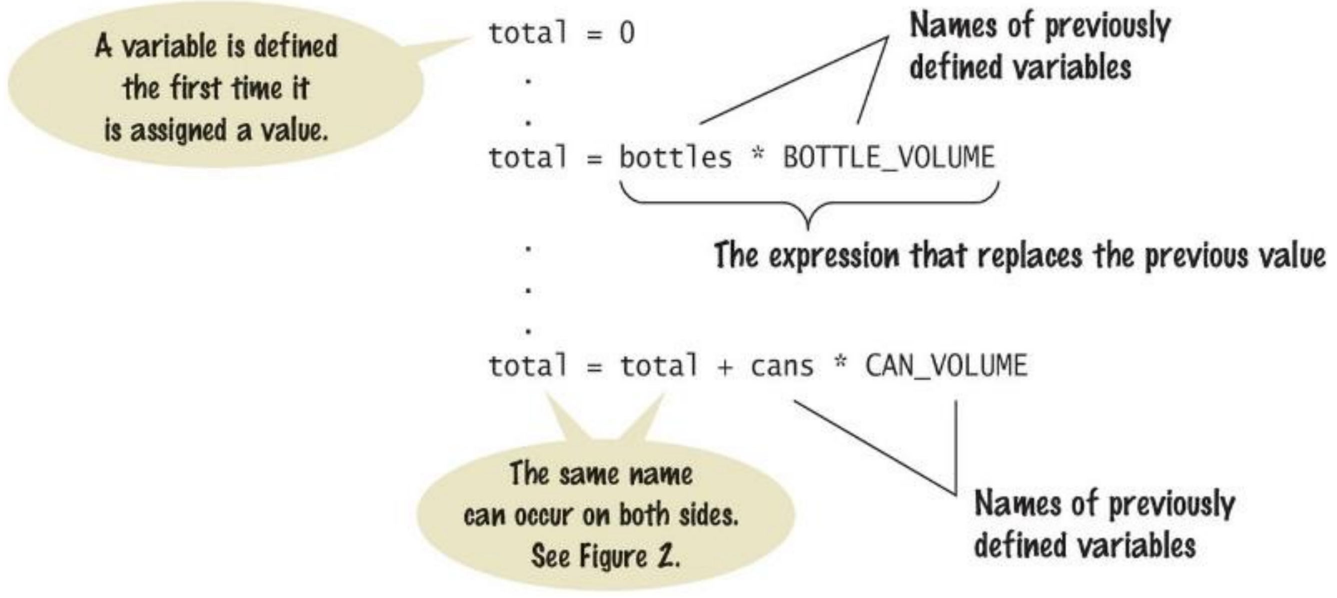
Variables

Variables

- A variable is a named storage location in a computer program
- There are many different types of variables, each type used to store different things
- You 'define' a variable by telling the compiler:
  - What name you will use to refer to it
  - The initial value of the variable
- You use an assignment statement to place a value into a variable

Variable Definition: assignment syntax

- To define a variable, you must specify an initial value.
- The value on the right of the '=' sign is assigned to the variable on the left


A variable is defined the first time it is assigned a value.
Names of previously defined variables
The expression that replaces the previous value
The same name can occur on both sides. See Figure 2.
Names of previously defined variables

```
total = 0
  .
  .
total = bottles * BOTTLE_VOLUME
  .
  .
  .
total = total + cans * CAN_VOLUME
```

The assignment statement

- Use the **assignment statement** '=' to place a new value into a variable

cansPerPack = 6   # define & initializes the variable cansPerPack

- Beware: The "=" sign is NOT used for comparison:
  - It copies the value on the right side into the variable on the left side
  - You will learn about the comparison operator later on in this course

Undefined Variables

- You must define a variable before you use it: (i.e. it must be defined somewhere above the line of code where you first use the variable)

```
canVolume = 12 * literPerOunce
literPerOunce = 0.0296
```

- The correct order for the statements is:

```
literPerOunce = 0.0296
canVolume = 12 * literPerOunce
```

An example: soda deal (volume1.py)

- Soft drinks are sold in cans and bottles. A store offers a six-pack of 12-ounce cans for the same price as a two-liter bottle. Which should you buy? (since 1 ounce is about 0,0296 liter, 12 fluid ounces equal approximately 0.355 liters.)

List of variables:	Type of Number
Number of cans per pack	Integer
Volume of a Bottle	Integer
Once per liter	float
Volume of a can	float
Volume of a 6-pax cans??	

Why different types?

There are three different types for data in python:

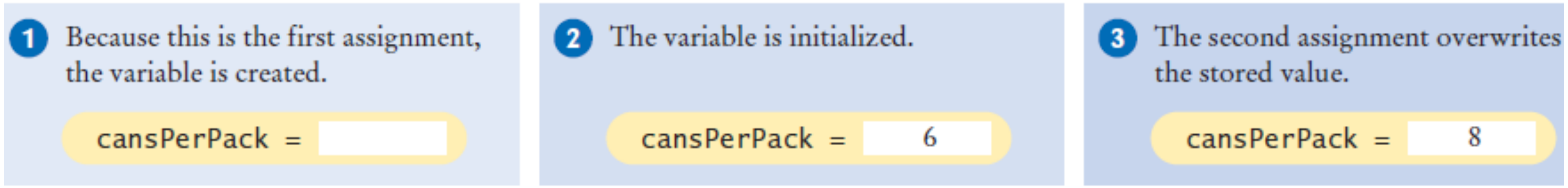
1.	A whole number (no fractional part)	7 (integer or int)
2.	A number with a fraction part	8.88 (float)
3.	A sequence of characters	"Bob" (string)
4.	A Boolean value	TRUE/FALSE (reserved keywords)

- The data type is associated with the **value**, not the **variable**:

```
cansPerPack = 6     # int
canVolume = 12.0   # float
```

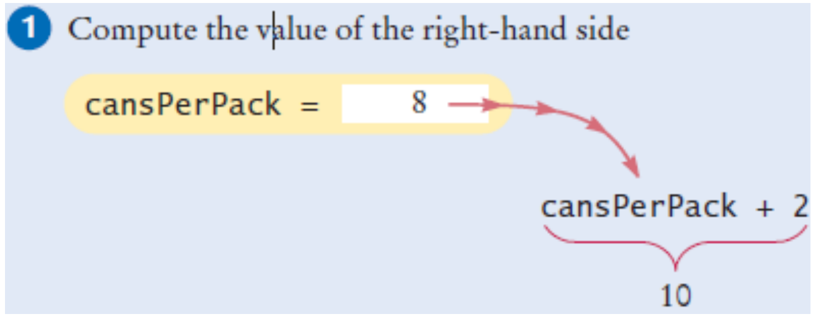
Updating a Variable (assigning a value)

- If an existing variable is assigned a new value, that value replaces the previous contents of the variable.
- For example:
  - cansPerPack = 6 ① ②
  - cansPerPack = 8 ③


1 Because this is the first assignment, the variable is created.
cansPerPack =
2 The variable is initialized.
cansPerPack = 6
3 The second assignment overwrites the stored value.
cansPerPack = 8

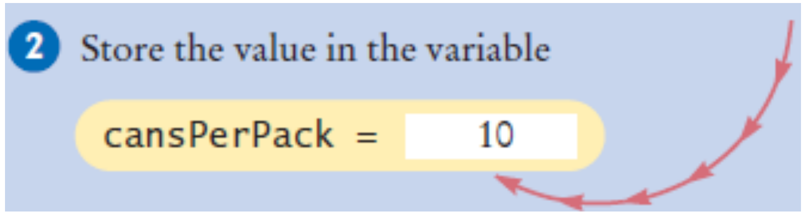
Updating a Variable (computed)

- Executing the Assignment:
  cansPerPack = cansPerPack + 2



Step by Step:

- Step 1: Calculate the right hand side of the assignment. Find the value of cansPerPack, and add 2 to it.
- Step 2: Store the result in the variable named on the left side of the assignment operator



Exercise: testing python types

- Open PyCharm (our IDE) and create a new file
  - type in the following
  - save the file as typetest.py
  - Run the program

```
# Testing different types in the same variable
taxRate = 5  # int
print(taxRate)
taxRate = 5.5  # float
print(taxRate)
taxRate = "Non-taxable" # string
print(taxRate)
print(taxRate + 5)
```

- So…
  - Once you have initialized a variable with a value of a particular type you should take great care to keep storing values of the same type in the variable

A Warning…

- Since the data type is associated with the value and not the variable:
    - A variable can be assigned different values at different places in a program

```
taxRate = 5                  # an int
```

- Then later…

```
taxRate = 5.5                # a float
```

- And then

```
taxRate = “Non- taxable” # a string
```

- If you use a variable and it has an unexpected type an error will occur in your program

A Minor Change

- Change line 8 to read:

```
print(taxRate + “??”)
```

- Save your changes
- Run the program
- What is the result?
- When you use the "+" operator with strings the second argument is concatenated to the end of the first
    - We'll cover string operations in more detail later in this chapter

Table 1: Number Literals in Python

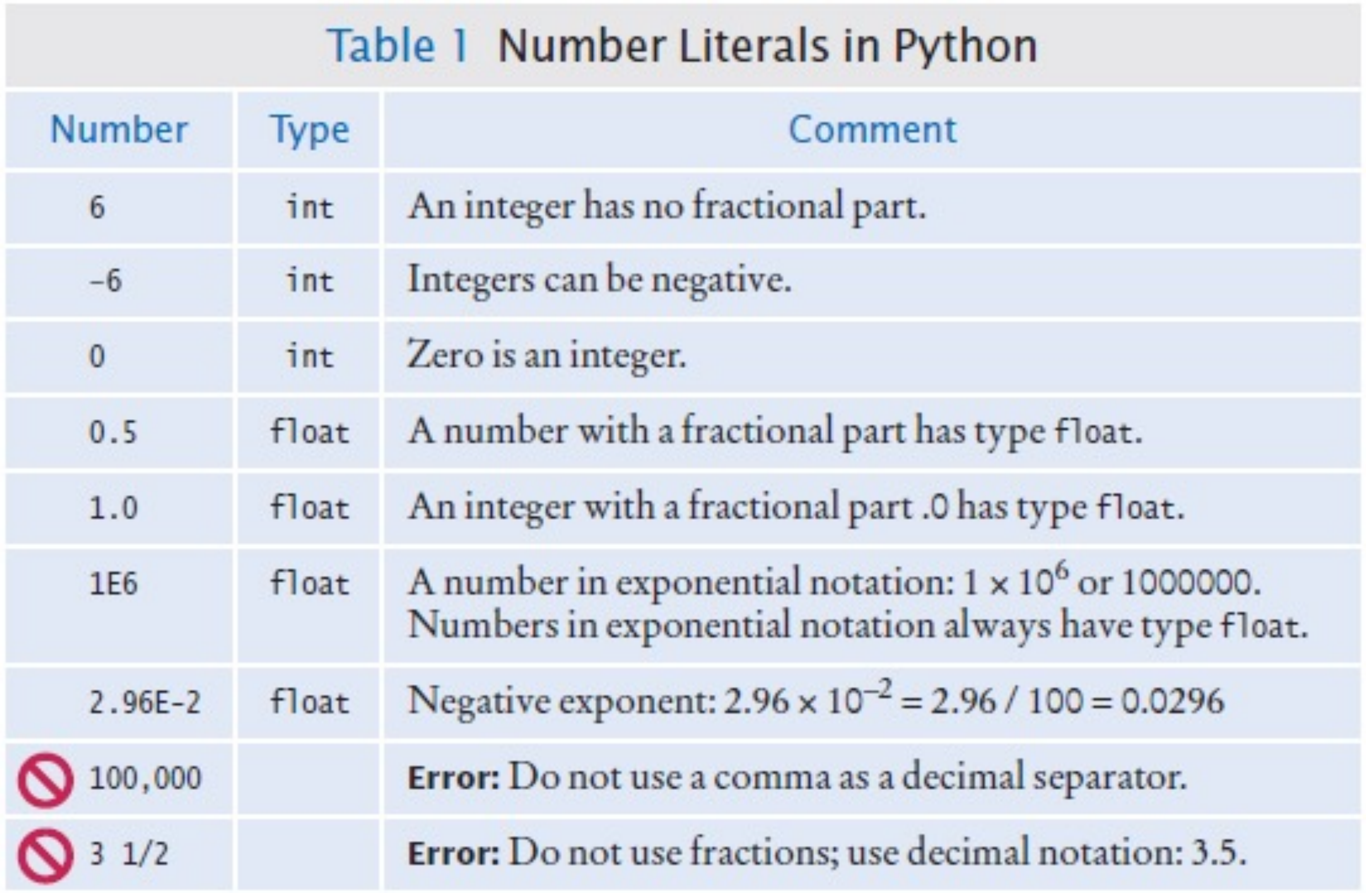


Table 1 Number Literals in Python

Number	Type	Comment
`6`	`int`	An integer has no fractional part.
`-6`	`int`	Integers can be negative.
`0`	`int`	Zero is an integer.
`0.5`	`float`	A number with a fractional part has type `float`.
`1.0`	`float`	An integer with a fractional part .0 has type `float`.
`1E6`	`float`	A number in exponential notation: 1×10^6 or 1000000. Numbers in exponential notation always have type `float`.
`2.96E-2`	`float`	Negative exponent: $2.96 \times 10^{-2} = 2.96 / 100 = 0.0296$
🚫 `100,000`		**Error:** Do not use a comma as a decimal separator.
🚫 `3 1/2`		**Error:** Do not use fractions; use decimal notation: 3.5.

Types of Literals in Python

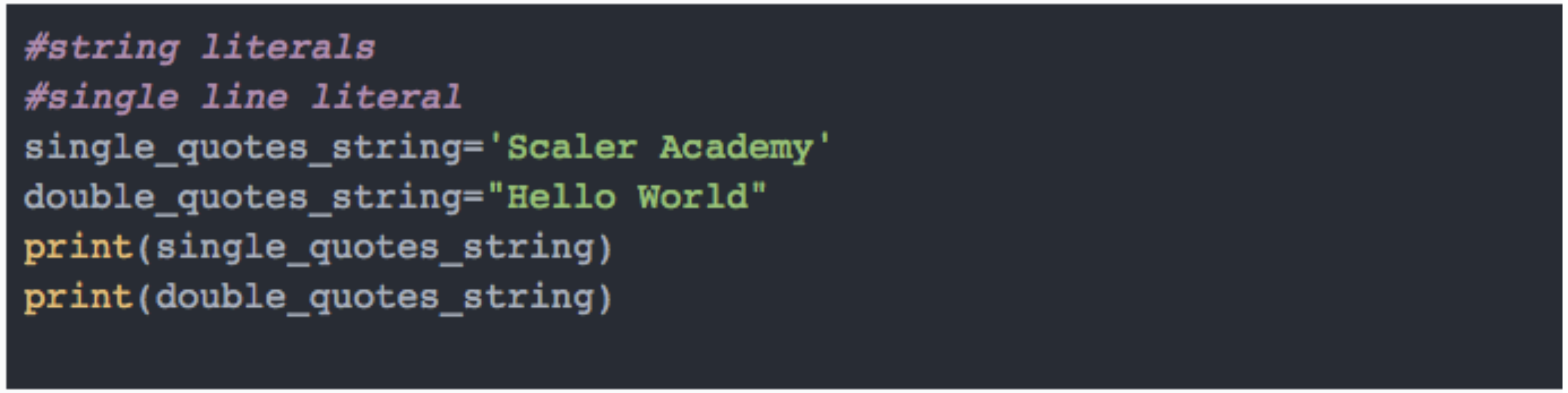
Python literals are of several types, and their usage is also pretty varied. So let's check them out one by one.

There are five types of literal in Python, which are as follows:

- String Literals
- Numeric Literals
- Boolean Literals
- Literal Collections
- Special Literals

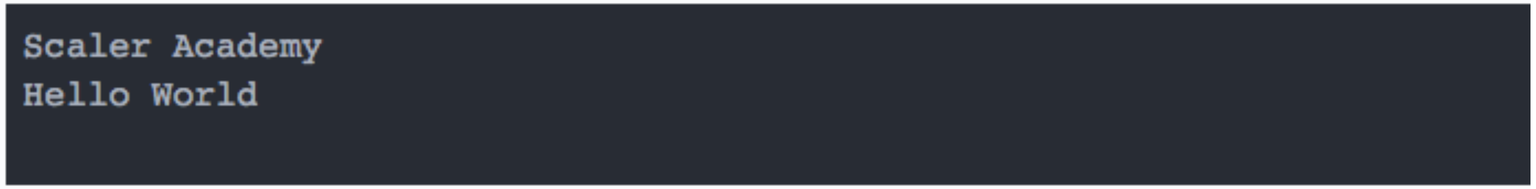
1. String Literals in Python

- Creating a string literal in Python is really easy- enclose the text or the group of characters in double quotes.



```
#string literals
#single line literal
single_quotes_string='Scaler Academy'
double_quotes_string="Hello World"
print(single_quotes_string)
print(double_quotes_string)
```

- We have successfully created a string using single quotes and double quotes in the above code snippet. Terminal



```
Scaler Academy
Hello World
```

2. Numeric Literals in Python

- Numerical literals in Python are those literals that contain digits only and are immutable. They are of four types:

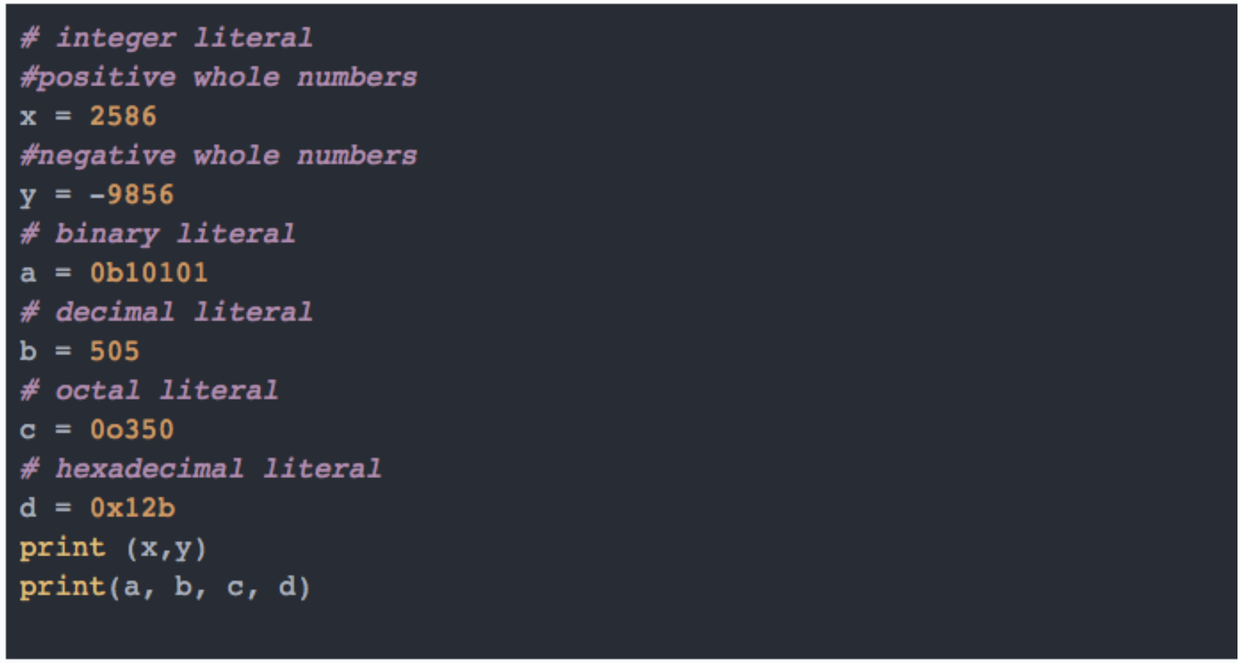
1. Integer

The numerical literals that are zero, positive or negative natural numbers and contain no decimal points are integers.

The different types of integers are:

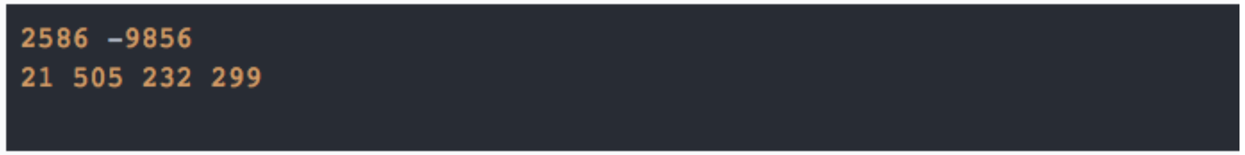
- **Decimal:** It contains digits from 0 to 9. The base for decimal values is 10.
- **Binary:** It contains only two digits- 0 and 1. The base for binary values is 2 and prefixed with "0b".
- **Octal**: It contains the digits from 0 to 7. The base for octal values is 8. In Python, such values are prefixed with "0o".
- **Hexadecimal**: It contains digits from 0 to 9 and alphabets from A to F.

Integer Examples:



```python
# integer literal
#positive whole numbers
x = 2586
#negative whole numbers
y = -9856
# binary literal
a = 0b10101
# decimal literal
b = 505
# octal literal
c = 0o350
# hexadecimal literal
d = 0x12b
print (x,y)
print(a, b, c, d)
```

- Check out the output for this

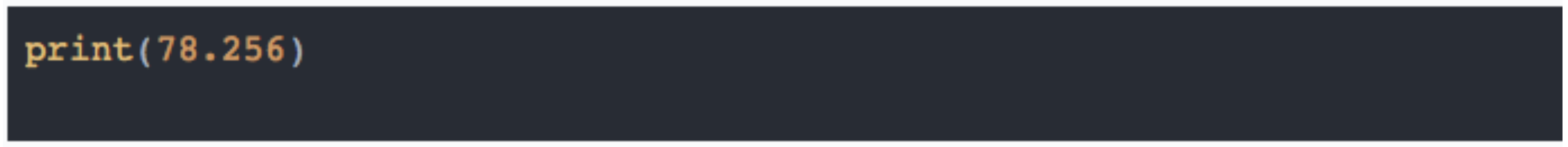


```
2586 -9856
21 505 232 299
```

2. Float

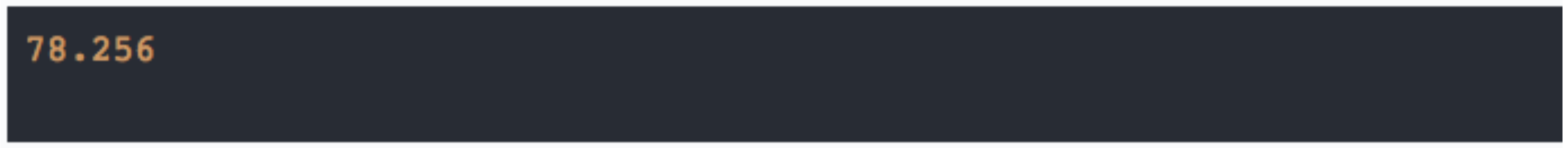
The floating-point literals are also known as real literals. Unlike integers, these contain decimal points. Float literals are primarily of two types:

- Fractional: Fractional literals contain both whole numbers and decimal points.
- An example of fractional literals will look like this:



```
print(78.256)
```

- Output

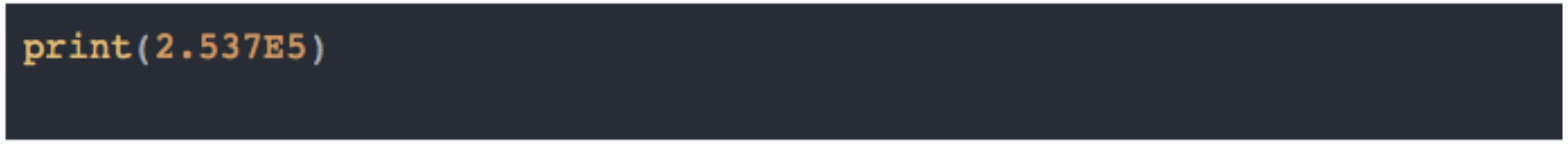


```
78.256
```

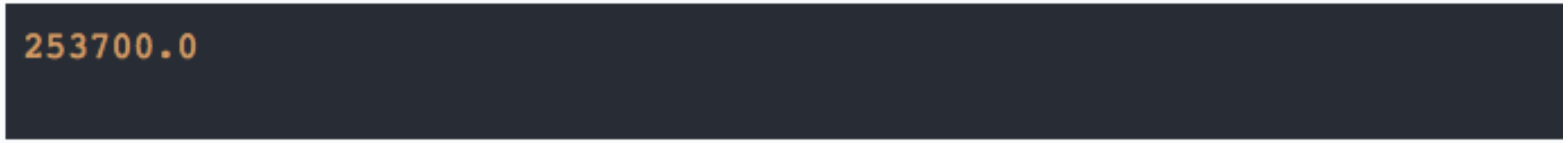
- Exponential: Exponential **literals in Python** are represented in the powers of 10. The power of 10 is represented by e or E. An exponential literal has two parts- the mantissa and the exponent.

  Note:

  - Mantissa: The digits before the symbol E in an exponential literal is known as the mantissa. In computing, it denotes the significant digits of the floating-point numbers.
  - Exponent: The digits after the symbol E in an exponential literal are the exponent. It denotes where the decimal point should be placed.

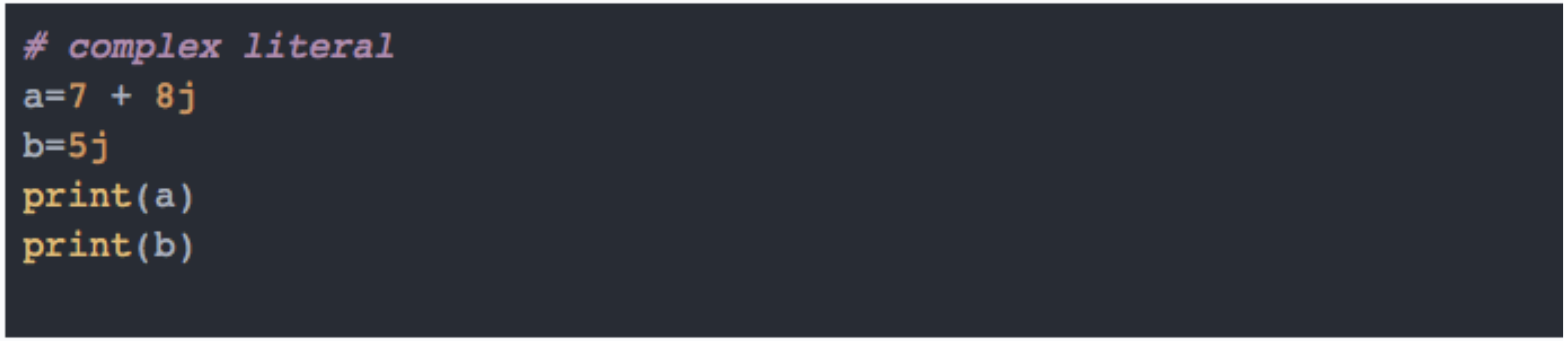


```
print(2.537E5)
```



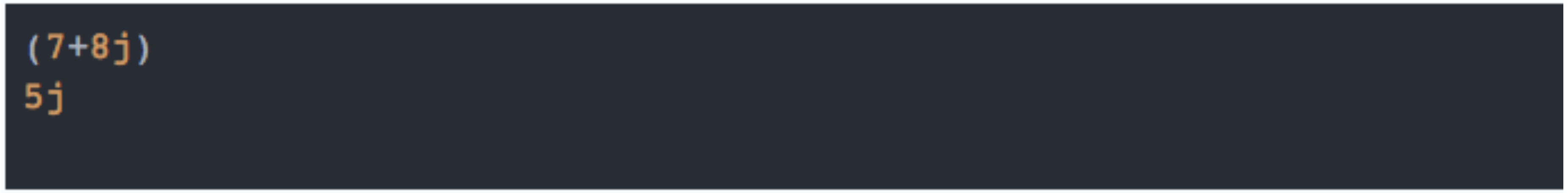
```
253700.0
```

- Complex: Complex literals are represented by A+Bj. Over here, A is the real part. And the entire B part, along with j, is the imaginary or complex part. j here represents the square root of -1, which is nothing but the iota or i we use in Mathematics.



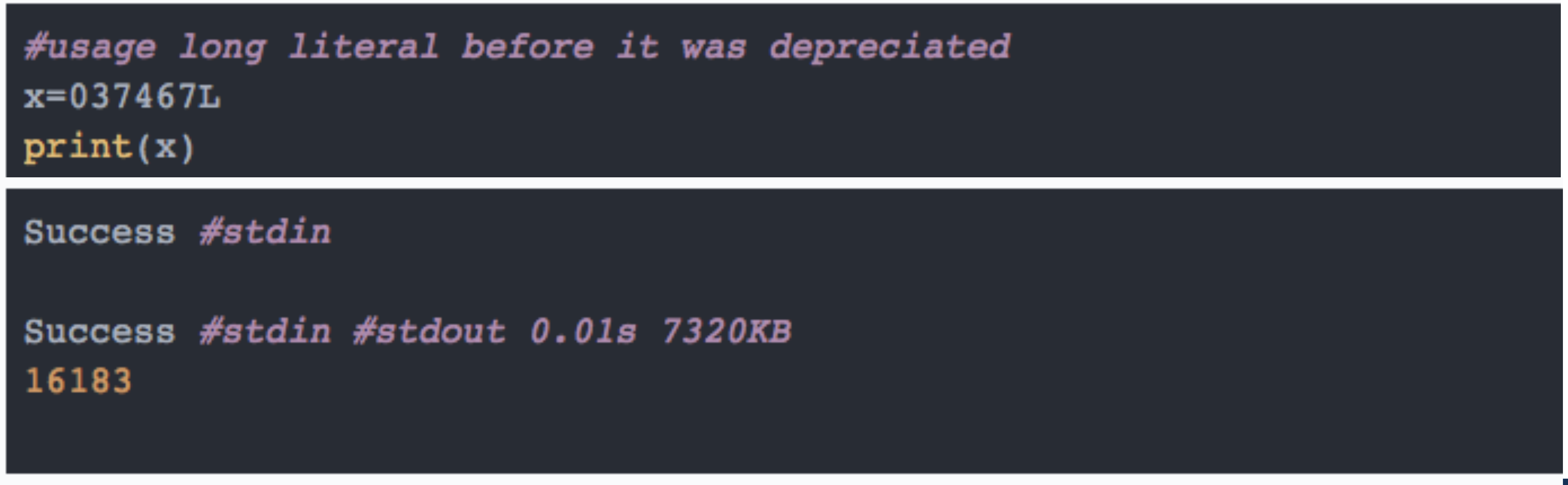
```python
# complex literal
a=7 + 8j
b=5j
print(a)
print(b)
```

- The output of the code snippet will be:



```
(7+8j)
5j
```

- Long: Long literals were nothing but integers with unlimited length. From Python 2.2 and onwards, the integers that used to overflow were automatically converted into long ints. Since Python 3.0, the long literal has been dropped. What was the long data type in Python 2 is now the standard int type in Python 3.
- Long literals used to be represented with a suffix- l or L. The usage of L was strongly recommended as l looked a lot like the digit 1. (executed using Python 1.8.)



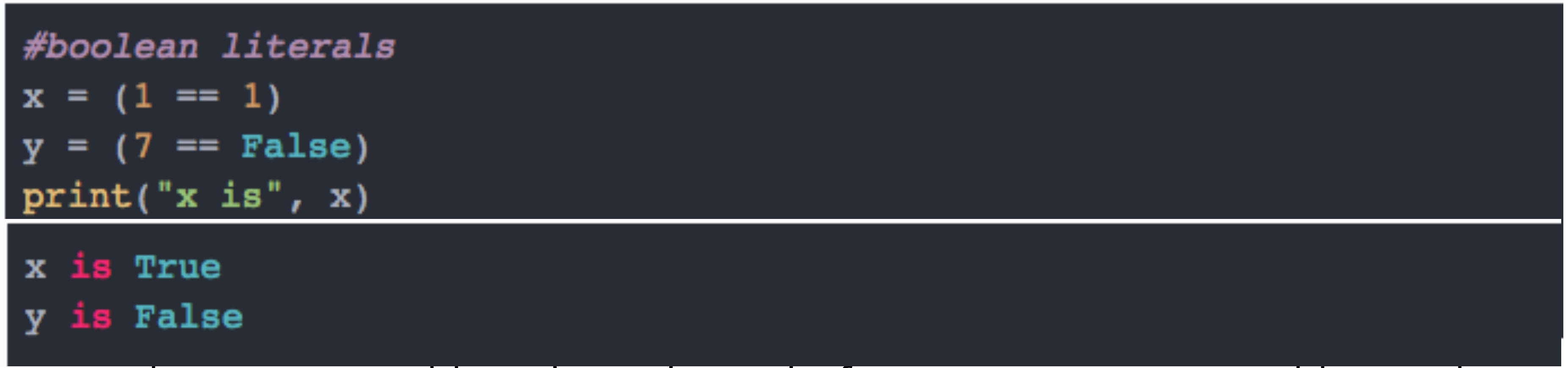
```
#usage long literal before it was depreciated
x=037467L
print(x)
```

```
Success #stdin

Success #stdin #stdout 0.01s 7320KB
16183
```

3. Boolean Literals in Python

- Boolean literals in Python are pretty straight-forward and have only two values:
  - True: True represents the value 1.
  - False: False represents the value 0.
- Let's see an example:



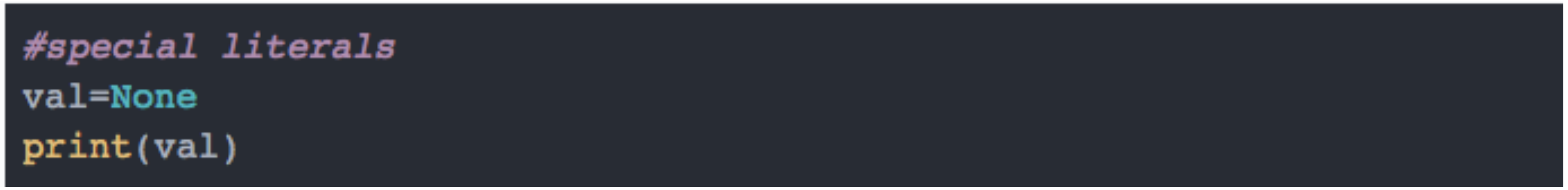
```
#boolean literals
x = (1 == 1)
y = (7 == False)
print("x is", x)
```

```
x is True
y is False
```

- We can see that we used boolean literals for comparison and based on the conditions, we received the outputs True and False, respectively.

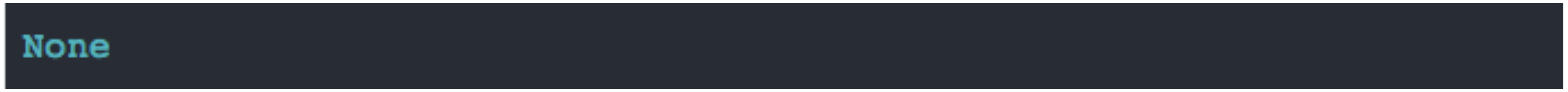
4. Special Literals in Python

- Python literals have one special literal known as None. This literal in Python is used to signify that a particular field is not created.
- Python will print None as output when we print the variable with no value assigned to it. None is also used for end of lists in Python.
- Example



```
#special literals
val=None
print(val)
```

- Output



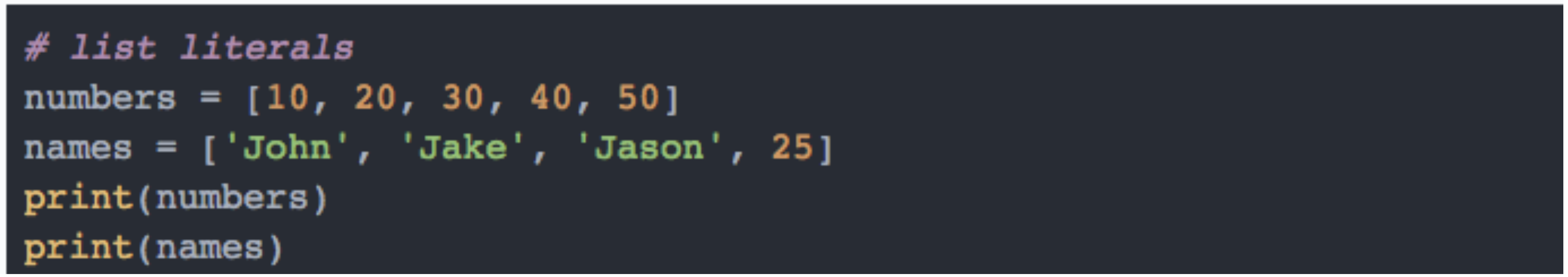
```
None
```

5. Literals Collections in Python

- If we wish to work with more than one value, then we can go for literal collections in Python.
- Literal collections in Python are of four types:
    1. List
    2. Tuple
    3. Dictionary
    4. Set

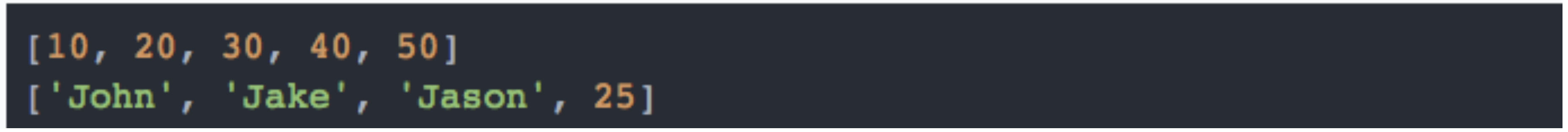
1. List

- Lists are a collection of data declared using the square brackets([]), and commas separate the elements of the list (,). This data can be of different types. Another important thing to know about lists is that they are mutable.
- Now let’s see an implementation:



```python
# list literals
numbers = [10, 20, 30, 40, 50]
names = ['John', 'Jake', 'Jason', 25]
print(numbers)
print(names)
```

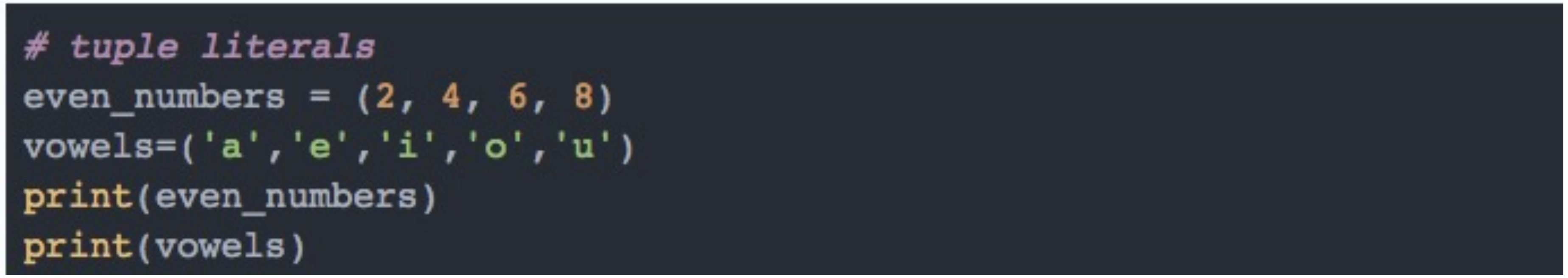
- Output



```
[10, 20, 30, 40, 50]
['John', 'Jake', 'Jason', 25]
```

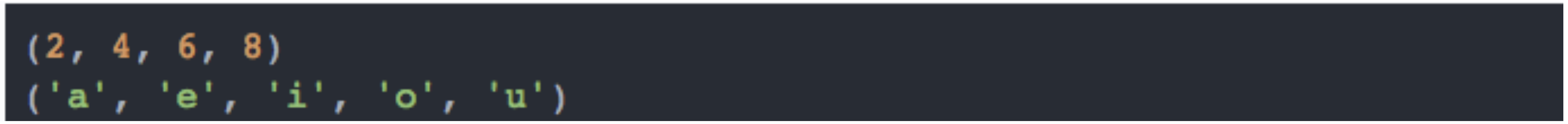
2. Tuple

- The literals that are declared using round brackets and can hold any data type are tuples. Commas separate the elements of tuples. However, unlike lists, tuples are immutable.
- Let's check out a code snippet on tuples:



```
# tuple literals
even_numbers = (2, 4, 6, 8)
vowels=('a','e','i','o','u')
print(even_numbers)
print(vowels)
```

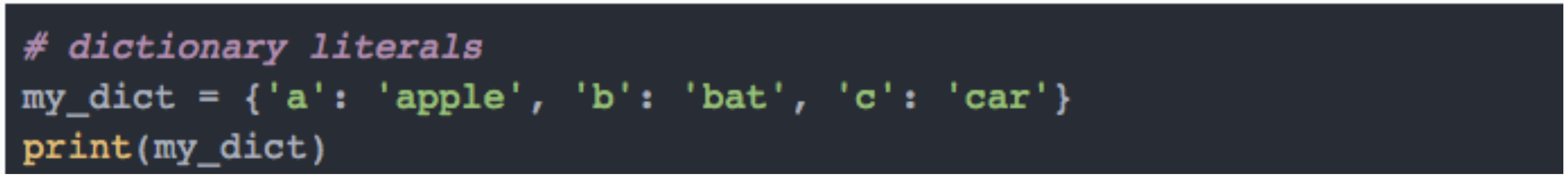
- Output



```
(2, 4, 6, 8)
('a', 'e', 'i', 'o', 'u')
```

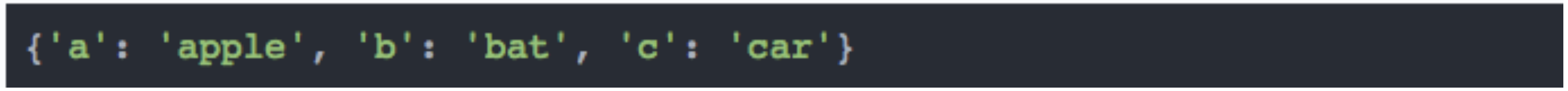
3. Dictionary

- Dictionary is a collection of data that stores value in a key-value format. These are enclosed in curly brackets and separated by commas. Dictionaries are mutable and can also contain different types of data.
- Check out the below code snippet that shows how a dictionary works



```
# dictionary literals
my_dict = {'a': 'apple', 'b': 'bat', 'c': 'car'}
print(my_dict)
```

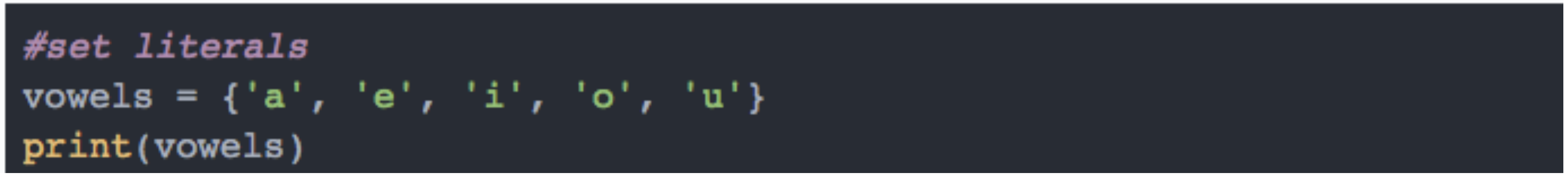
- Output



```
{'a': 'apple', 'b': 'bat', 'c': 'car'}
```

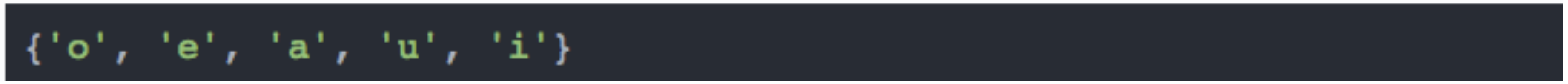
4. Set

- Set literals are a collection of unordered data that cannot be modified. It is enclosed within curly brackets and separated by commas.
- Let's see the code for this:



```python
#set literals
vowels = {'a', 'e', 'i', 'o', 'u'}
print(vowels)
```

- Output



```
{'o', 'e', 'a', 'u', 'i'}
```

Naming variables

- Variable names should describe the purpose of the variable
  - 'canVolume' is better than 'cv'
- Use These Simple Rules
  1. Variable names must start with a letter or the underscore (_) character
     1. Continue with letters (upper or lower case), digits or the underscore
  2. You cannot use other symbols (? or %...) and spaces are not permitted
  3. Separate words with 'camelCase' notation
     1. Use upper case letters to signify word boundaries
  4. Don't use 'reserved' Python words (see Appendix C, pages A6 and A7)

Table 2: Variable Names in Python

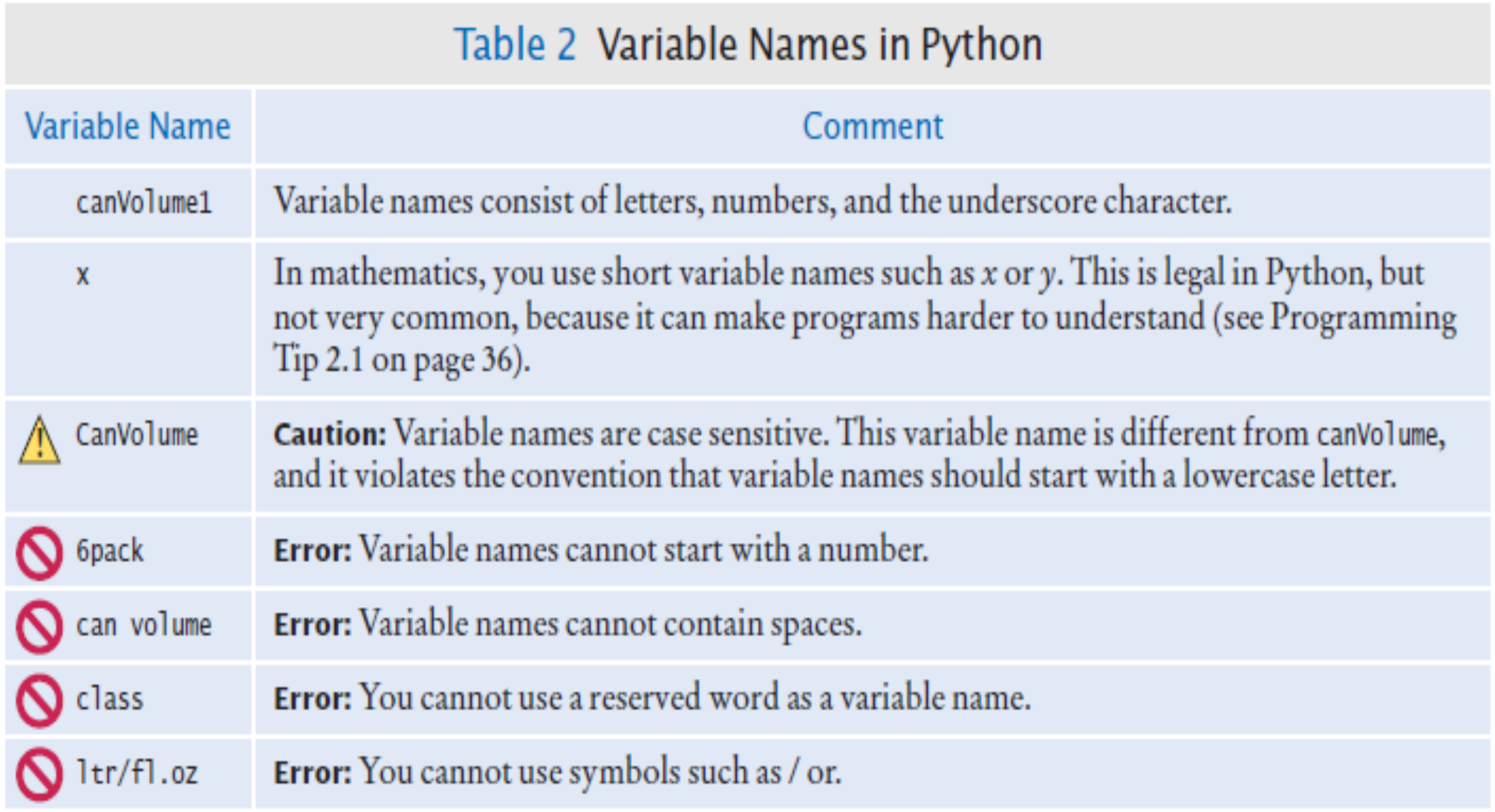


Table 2 Variable Names in Python

Variable Name	Comment
`canVolume1`	Variable names consist of letters, numbers, and the underscore character.
`x`	In mathematics, you use short variable names such as *x* or *y*. This is legal in Python, but not very common, because it can make programs harder to understand (see Programming Tip 2.1 on page 36).
⚠ `CanVolume`	**Caution:** Variable names are case sensitive. This variable name is different from `canVolume`, and it violates the convention that variable names should start with a lowercase letter.
🚫 `6pack`	**Error:** Variable names cannot start with a number.
🚫 `can volume`	**Error:** Variable names cannot contain spaces.
🚫 `class`	**Error:** You cannot use a reserved word as a variable name.
🚫 `ltr/fl.oz`	**Error:** You cannot use symbols such as / or.

Programming Tip: Use Descriptive Variable Names

- Choose descriptive variable names
- Which variable name is more self descriptive?

```
canVolume = 0.35
cv = 0.355
```

- This is particularly important when programs are written by more than one person.

Constants: Naming & Style

- In Python a **constant** is a variable whose value ***should not*** be changed after it's assigned an initial value.
  - It is a good practice to use all UPPER_CASE when naming constants

```
BOTTLE_VOLUME = 2.0
```

- It is good style to use named constants to explain numerical values to be used in calculations
  - Which is clearer?

```
totalVolume = bottles * 2

totalVolume = bottles * BOTTLE_VOLUME
```

- A programmer reading the first statement may not understand the significance of the "2"
- Python will let you change the value of a **constant**
  - Just because you can do it, doesn't mean you should do it

Constants: Naming & Style

- It is customary to use all UPPER_CASE letters for constants to distinguish them from variables.
  - It is a nice visual way cue

```
BOTTLE_VOLUME = 2    # Constant
MAX_SIZE = 100       # Constant
taxRate = 5          # Variable
```

Programming Tips: Python comments

- Use comments at the beginning of each program to clarify details of the code
- Comments are a courtesy to others and a way to document your thinking
    - Comments to add explanations for humans who read your code.
- The compiler ignores comments.

Arithmetic

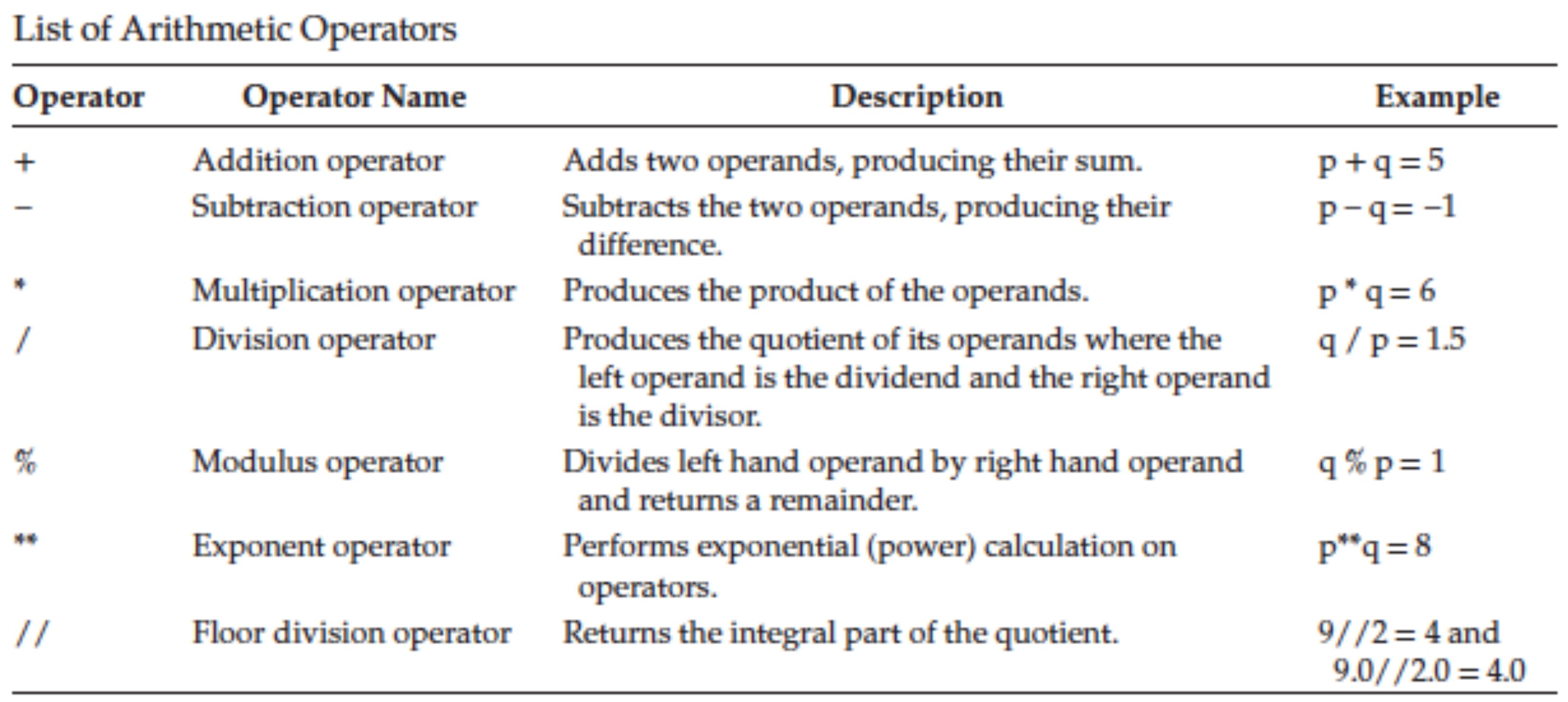
Basic Arithmetic Operations

- Python supports all of the basic arithmetic operations:
  - Addition "+"
  - Subtraction "-"
  - Multiplication "*"
  - Division "/"
- You write your expressions a bit differently

$$\frac{a+b}{2}$$

```
(a + b) / 2
```

List of Arithmetic Operators



List of Arithmetic Operators

Operator	Operator Name	Description	Example
+	Addition operator	Adds two operands, producing their sum.	p + q = 5
–	Subtraction operator	Subtracts the two operands, producing their difference.	p − q = −1
*	Multiplication operator	Produces the product of the operands.	p * q = 6
/	Division operator	Produces the quotient of its operands where the left operand is the dividend and the right operand is the divisor.	q / p = 1.5
%	Modulus operator	Divides left hand operand by right hand operand and returns a remainder.	q % p = 1
**	Exponent operator	Performs exponential (power) calculation on operators.	p**q = 8
//	Floor division operator	Returns the integral part of the quotient.	9//2 = 4 and 9.0//2.0 = 4.0

Precedence

- Precedence is similar to Algebra:
  - PEMDAS
    - Parenthesis, Exponent, Multiply/Divide, Add/Subtract

Unbalanced Parentheses

- Consider the expression
  `((a + b) * t / 2 * (1 - t)`
  - What is wrong with the expression?
- Now consider this expression.
  `(a + b) * t) / (2 * (1 - t)`
  - This expression has three "(" and three ")", but it still is not correct
- At any point in an expression the count of "(" must be greater than or equal to the count of ")"
- At the end of the expression the two counts must be the same

Powers

- Double stars ** are used to calculate an exponent
- Analyzing the expression: $b \times \left(1 + \frac{r}{100}\right)^n$
- Convert this expression in Python language

Powers

- Double stars ** are used to calculate an exponent
- Analyzing the expression: $b \times \left(1 + \frac{r}{100}\right)^n$
- Becomes:
  - b * ((1 + r / 100) ** n)

```
b * (1 + r / 100) ** n
```

$$\underbrace{\texttt{b *} \underbrace{\texttt{(}\underbrace{\texttt{1 + } \underbrace{\texttt{r / 100}}_{\frac{r}{100}}}_{1+\frac{r}{100}}\texttt{) ** n}}_{\left(1+\frac{r}{100}\right)^{n}}}_{b\times\left(1+\frac{r}{100}\right)^{n}}$$

Testing Operators: test_operators.py

- Write a python program to test all the arithmetic operators.

Mixing numeric types

- If you mix integer and floating-point values in an arithmetic expression, the result is a floating-point value.
- 7 + 4.0   # Yields the floating value 11.0
- Remember from our earlier example:
  - If you mix stings with integer or floating point values the result is an error

Floor division

- When you divide two integers with the / operator, you get a floating-point value. For example,

```
7 / 4
```

- Yields 1.75
- We can also perform **floor division** using the // operator.
  - The "//" operator computes the quotient and discards the fractional part

```
7 // 4
```

- Evaluates to 1 because 7 divided by 4 is 1.75 with a fractional part of 0.75, which is discarded.

Calculating a remainder

- If you are interested in the remainder of dividing two integers, use the "%" operator (called modulus):

```
remainder = 7 % 4
```

- The value of remainder will be 3
- Sometimes called modulo divide

A Simple Example: pennies_dollars_conversion.py

- Open a new file in the PyCharm IDE:
- Type in the following:

```
# Convert pennies to dollars and cents (1 penny is equal to 0.01 dollar)
pennies = 1729                #it is just an example
dollars = ???                # Calculates the number of dollars
cents = ???                  # Calculates the number of pennies
print("I have", dollars, ”dollars and", cents, "cents")
```

- Save the file
- Run the file
- What is the result?

A Simple Example: pennies_dollars_conversion.py

- Open a new file in the PyCharm IDE:
- Type in the following:

```
# Convert pennies to dollars and cents (1 penny is equal to 0.01 dollar)
pennies = 1729
dollars = pennies // 100  # Calculates the number of dollars
cents = pennies % 100     # Calculates the number of pennies
print("I have", dollars, ”dollars and", cents, "cents")
```

- Save the file
- Run the file
- What is the result?

Integer Division and Remainder Examples

- Handy to use for making change:
  - pennies = 1729
  - dollars = pennies // 100 # 17
  - cents = pennies % 100   # 29

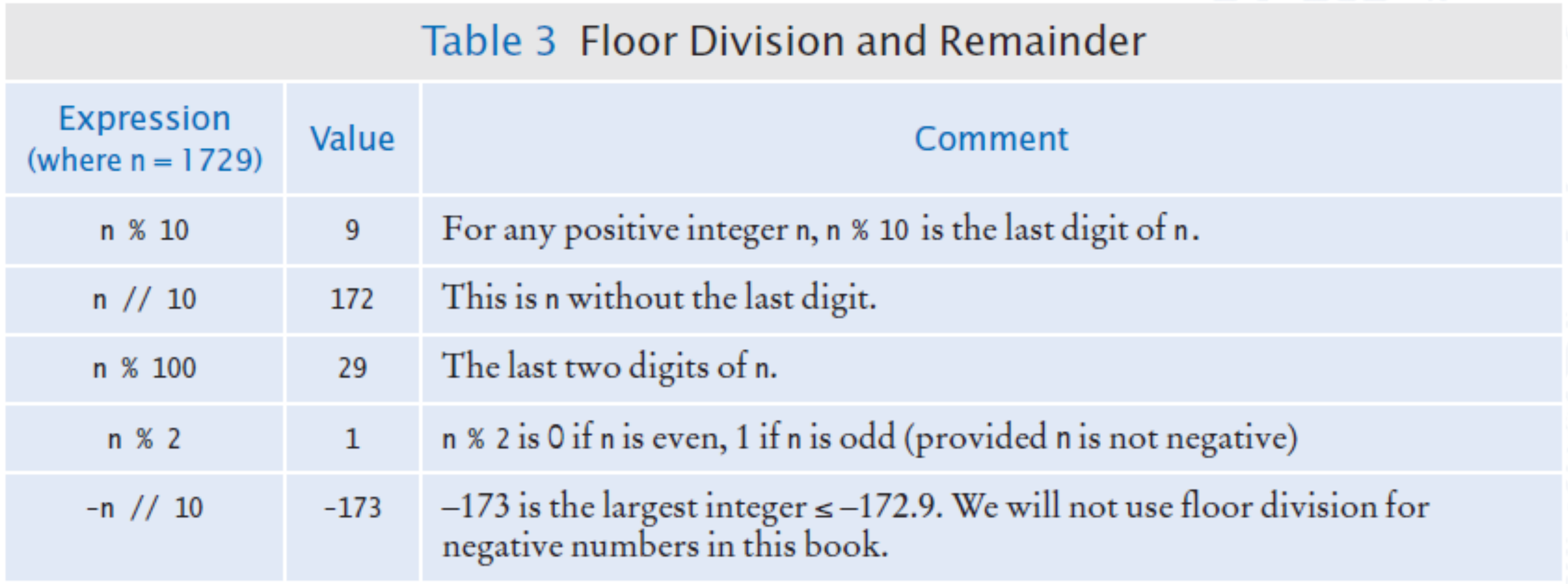


Table 3 Floor Division and Remainder

Expression (where n = 1729)	Value	Comment
`n % 10`	9	For any positive integer n, `n % 10` is the last digit of n.
`n // 10`	172	This is n without the last digit.
`n % 100`	29	The last two digits of n.
`n % 2`	1	`n % 2` is 0 if n is even, 1 if n is odd (provided n is not negative)
`-n // 10`	-173	–173 is the largest integer ≤ –172.9. We will not use floor division for negative numbers in this book.

Exercise: vending.py

- Write a program that simulates *a vending machine that gives change.*
- *Ask the user to insert the amount of Dollars he/she uses to pay:*

```
d= int(input('Insert coin:1 for 1$ and 5 for 5$: '))
```

- *Insert the price in pennies of the object the user wants to buy*
- *Return the change due expressed in dollars and quarters*
- *Consider that:*

PENNIES_PER_DOLLAR = 100
PENNIES_PER_QUARTER = 25

Floating-point to integer conversion

- You can use the function int() and float() to convert between integer and floating point values:

```
      balance = total + tax   # balance: float
      dollars = int (balance) # dollars: integer
      number = int(input("Enter an integer: ")) # input return a string,
but we want to read it as a integer  number
```

- You lose the fractional part of the floating-point value (no rounding occurs)

Built in Mathematical Functions

- **Built-in** functions are a small set of functions that are defined as a part of the Python language
  - They can be used without importing any modules

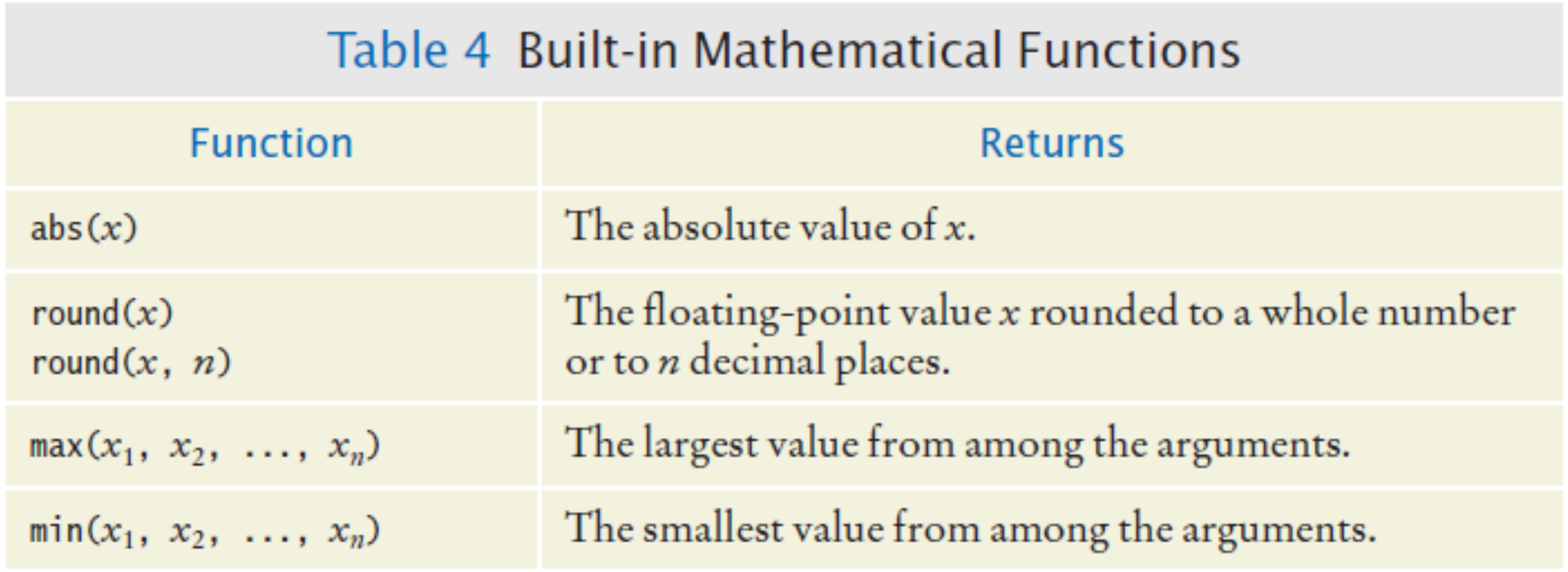


Table 4 Built-in Mathematical Functions

Function	Returns
`abs(`x`)`	The absolute value of x.

`round(`x`)`
`round(`x`, `n`)`
	The floating-point value x rounded to a whole number or to n decimal places.

`max(`$x_1, x_2, \ldots, x_n$`)`	The largest value from among the arguments.
`min(`$x_1, x_2, \ldots, x_n$`)`	The smallest value from among the arguments.

Calling functions

- Recall that a function is a collection of programming instructions that carry out a particular task.
- The print() function can display information, but there are many other functions available in Python.
- When calling a function you must provide the correct number of arguments
    - The program will generate an error message if you don’t

Calling functions that return a value

- Most functions return a value. That is, when the function completes its task, it passes a value back to the point where the function was called.
- For example:
  - The call abs(-173) returns the value 173.
  - The value returned by a function can be stored in a variable:
    - distance = abs(x)
- You can use a function call as an argument to the **print** function
- Go to the python shell window in Wing and type:

```
print(abs(-173))
```

Functions from the Math Module

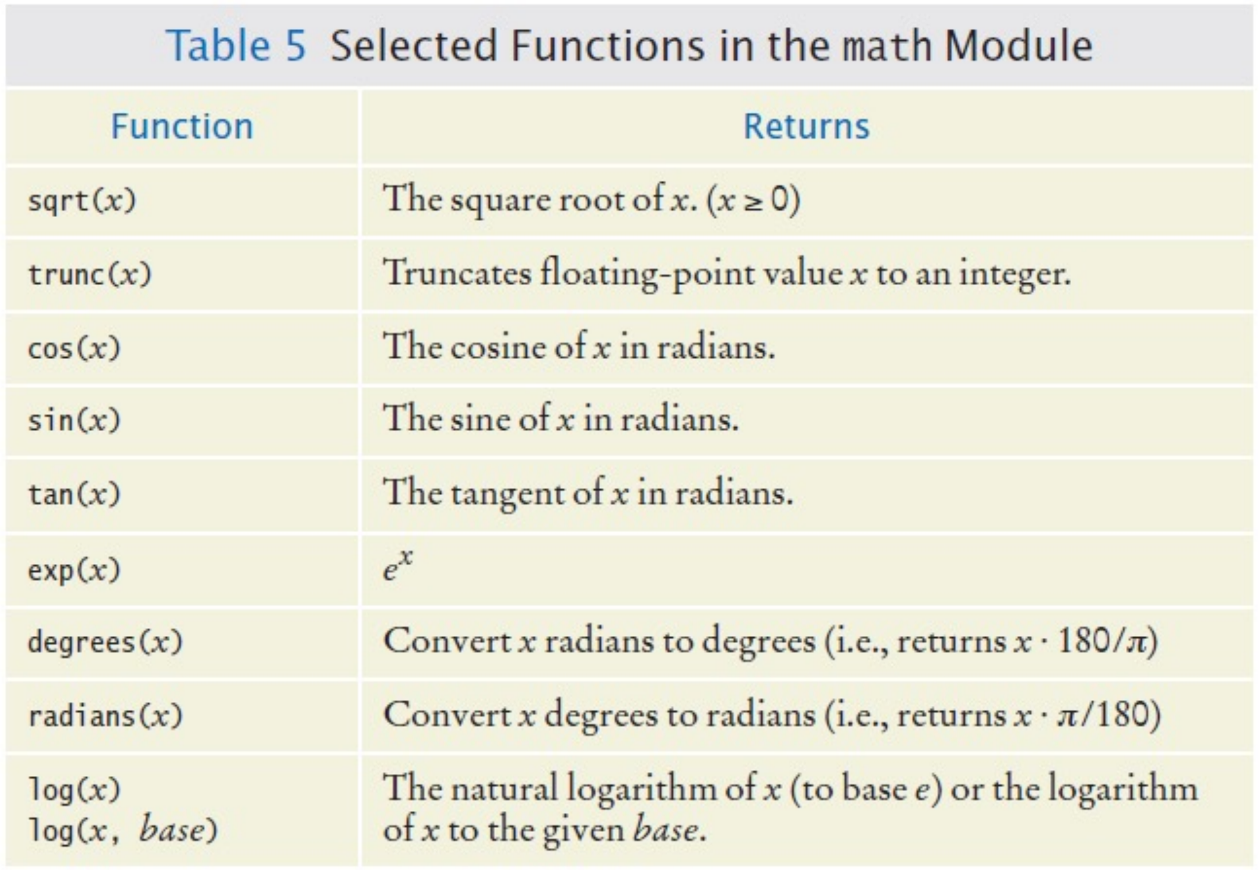


Table 5 Selected Functions in the math Module

Function	Returns
`sqrt(x)`	The square root of x. ($x \geq 0$)
`trunc(x)`	Truncates floating-point value x to an integer.
`cos(x)`	The cosine of x in radians.
`sin(x)`	The sine of x in radians.
`tan(x)`	The tangent of x in radians.
`exp(x)`	e^x
`degrees(x)`	Convert x radians to degrees (i.e., returns $x \cdot 180/\pi$)
`radians(x)`	Convert x degrees to radians (i.e., returns $x \cdot \pi/180$)

`log(x)`
`log(x, base)`
	The natural logarithm of x (to base e) or the logarithm of x to the given *base*.

Python libraries (modules)

- A **library** is a collection of code, written and compiled by someone else, that is ready for you to use in your program
- A **standard library** is a library that is considered part of the language and must be included with any Python system.
- Python's standard library is organized into **modules**.
  - Related functions and data types are grouped into the same module.
  - Functions defined in a module must be explicitly loaded into your program before they can be used.

Arithmetic Expressions: operators+functions

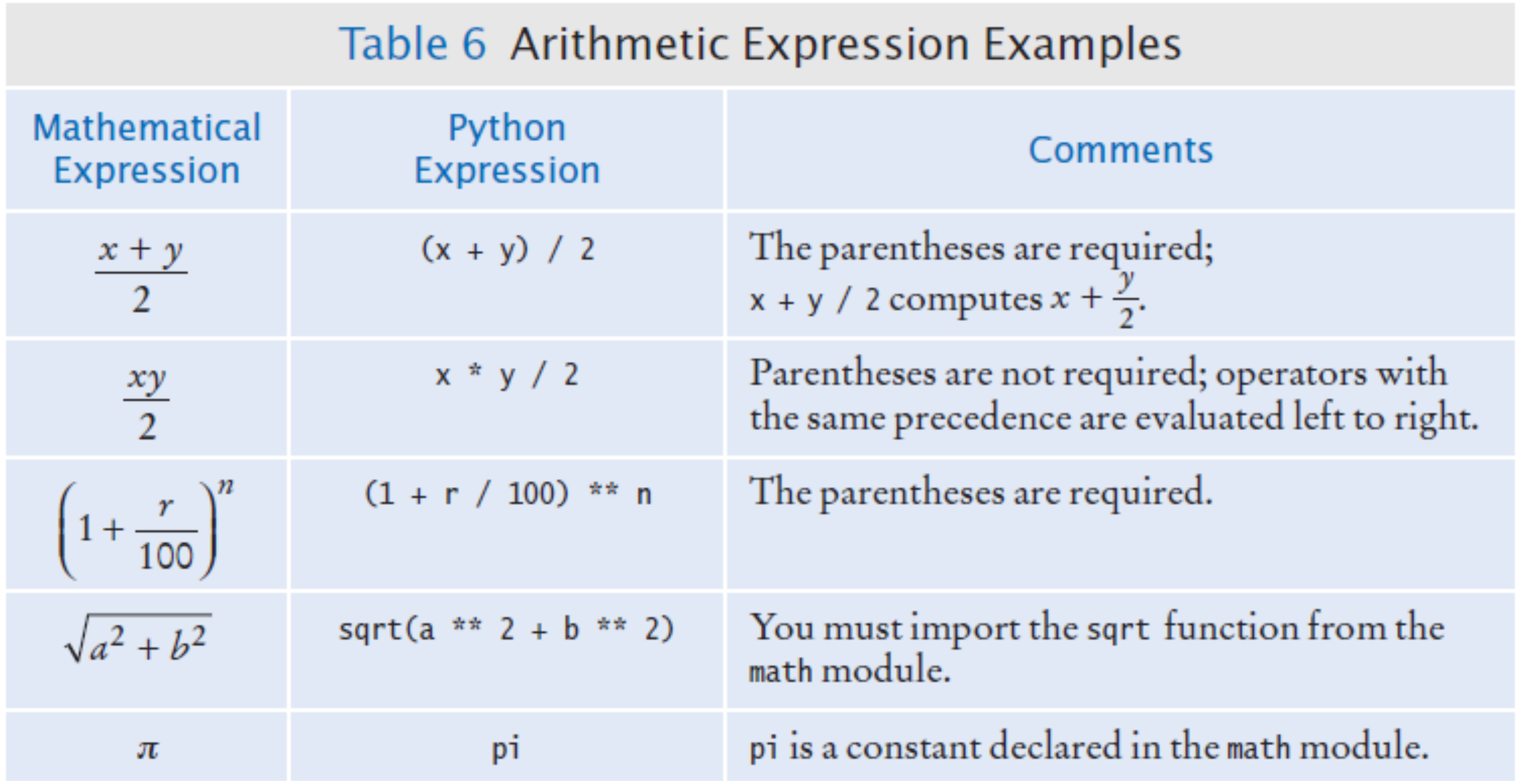


Table 6 Arithmetic Expression Examples

Mathematical Expression	Python Expression	Comments
$\frac{x+y}{2}$	`(x + y) / 2`	The parentheses are required; `x + y / 2` computes $x + \frac{y}{2}$.
$\frac{xy}{2}$	`x * y / 2`	Parentheses are not required; operators with the same precedence are evaluated left to right.
$\left(1+\frac{r}{100}\right)^n$	`(1 + r / 100) ** n`	The parentheses are required.
$\sqrt{a^2+b^2}$	`sqrt(a ** 2 + b ** 2)`	You must import the `sqrt` function from the `math` module.
π	`pi`	`pi` is a constant declared in the `math` module.

Using functions from the Math Module: math_function.py

- For example, to use the sqrt() function, which computes the square root of its argument:

```
# First include this statement at the top of your
# program file.
from math import sqrt
```

# Then you can simply call the function as

```
y = sqrt(x)
```

Excercise: arithmetic.py

Read two integers and display the result of:

- Sum
- Difference
- Product
- Average
- Distance
- Minimum
- Maximum

Produce a legible output (use formatting style to show data as in columns)

Excercise: arithmetic.py

Write in python the following mathematical expressions:

$$s = s_0 + v_0 t + \frac{1}{2} g t^2$$

$$G = 4\pi^2 \frac{a^3}{p^2(m_1 + m_2)}$$

$$FV = PV \cdot \left(1 + \frac{INT}{100}\right)^{YRS}$$

$$c = \sqrt{a^2 + b^2 - 2ab\cos\gamma}$$

Additional Programming Tips

- Use Spaces in expressions: `totalCans = fullCans + emptyCans`
- Is easier to read than: `totalCans=fullCans+emptyCans`
- Other ways to import modules:

```
from math import, sqrt, sin, cos  # imports the functions
from math import *  # imports all functions from the module
Import math # imports all functions from the module
```

- If you use the last style you have to add the module name and a "." before each function call: `y = math.sqrt(x)`

Problem Solving

Exercise : traveltime.py
Compute the shift time

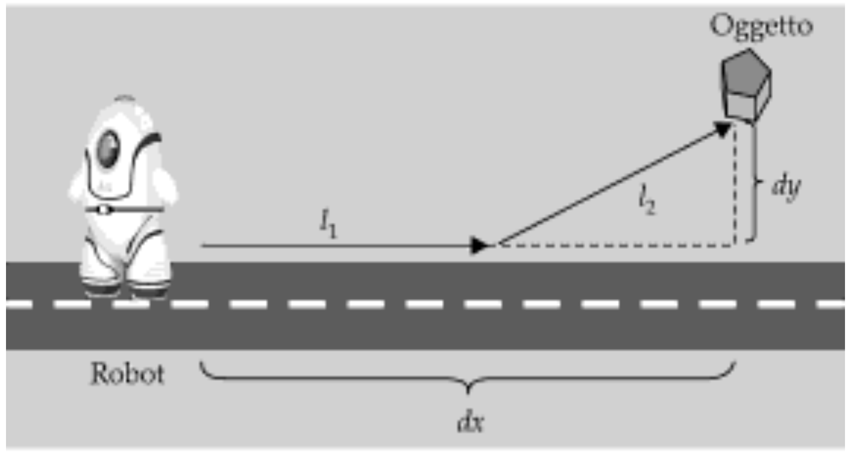
- A robot must retrieve an object that is located in a rocky terrain, next to a road.
- The robot can travel at a higher speed when it is on the road than it can do on rocky ground, so it will travel a certain stretch of road, then move in a straight line to the object.
- Calculate the total time it takes for the robot to achieve its goal.

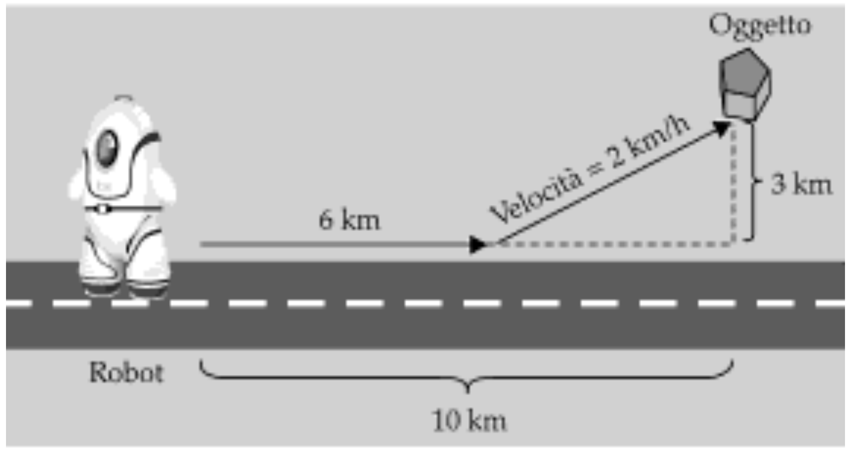
Exercise: Compute the shift time

We need to calculate the total time it takes for the robot to achieve its goal, based on the following input data:

- The distance between the robot and the object in the x direction and in the y direction (respectively, dx and dy).
- The speed of the robot on the road (s1) and on rocky terrain (s2).
- The length (l1) of the first section of the movement (which takes place on the road).

To give a concrete example of the problem, we use these values: dx=10km, dy=3km, s1=5km/h, s2=2km/h, l1=6km.





Strings

Strings

- Start with some simple definitions:
  - Text consists of **characters**
  - **Characters** are letters, numbers, punctuation marks, spaces, ….
  - A **string** is a sequence of **characters**
- In Python, string literals are specified by enclosing a sequence of **characters** within a matching pair of either single or double quotes.

```
print("This is a string.", 'So is this.')
```

- By allowing both types of delimiters, Python makes it easy to include an apostrophe or quotation mark within a string.
  - `message = 'He said "Hello"'`
  - Remember to use matching pairs of quotes, single with single, double with double

String Operations

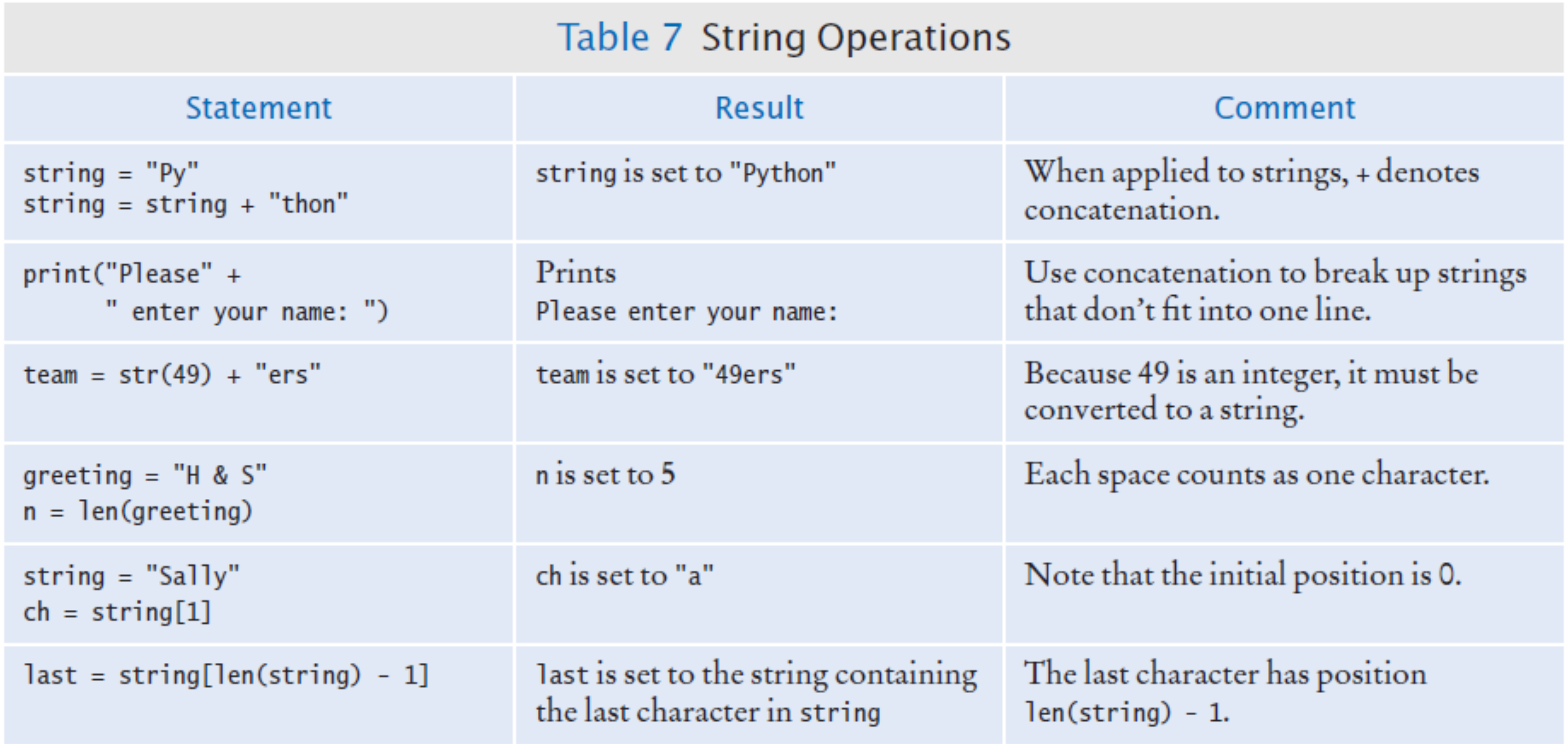


Table 7 String Operations

Statement	Result	Comment

`string = "Py"`
`string = string + "thon"`
	`string` is set to `"Python"`	When applied to strings, + denotes concatenation.

`print("Please" +`
`      " enter your name: ")`
	Prints
	`Please enter your name:`
		Use concatenation to break up strings that don't fit into one line.

`team = str(49) + "ers"`	`team` is set to `"49ers"`	Because 49 is an integer, it must be converted to a string.

`greeting = "H & S"`
`n = len(greeting)`
	`n` is set to 5	Each space counts as one character.

`string = "Sally"`
`ch = string[1]`
	`ch` is set to `"a"`	Note that the initial position is 0.

`last = string[len(string) - 1]`	`last` is set to the string containing the last character in `string`	The last character has position `len(string) - 1`.

String Length

- The number of characters in a string is called the length of the string. (For example, the length of "Harry" is 5).
- You can compute the length of a string using Python’s len() function:

```
length = len("World!") # length is 6
```

- A string of length 0 is called the empty string. It contains no characters and is written as "" or ''.

String Concatenation (“+”)

- You can ‘add’ one String onto the end of another

```
firstName = "Harry"
lastName = "Morgan"
name  = firstName + lastName  # HarryMorgan
print(“my name is:”, name)
```

- You wanted a space in between the two names?

```
name = firstName + " " + lastName  # Harry Morgan
```

Using “+” to concatenate strings is an example of a concept called operator overloading. The “+” operator performs different functions of variables of different types

String repetition (“*”)

- You can also produce a string that is the result of repeating a string multiple times.
- Suppose you need to print a dashed line.
- Instead of specifying a literal string with 50 dashes, you can use the * operator to create a string that is comprised of the string "-" repeated 50 times.

```
dashes = "-" * 50
```

- results in the string "--”

The “*” operator is also overloaded.

Converting Numbers to Strings

- Use the str() function to convert between numbers and strings.
- Open Wing, then open a new file and type in:

```
balance = 888.88
dollars = 888
balanceAsString = str(balance)
dollarsAsString = str(dollars)
print(balanceAsString)
print(dollarsAsString)
```

- To turn a string containing a number into a numerical value, we use the int() and float() functions:

```
id = int("1729")
price = float("17.29")
print(id)
print(price)
```

- **This conversion is important when the strings come from user input**.

Accessing a String

- Each char inside a String has an index number:

0	1	2	3	4	5	6	7	8	9
c	h	a	r	s		h	e	r	e

- The first char is index zero (0)
- The [] operator returns a char at a given index inside a String:

0	1	2	3	4
H	a	r	r	y

```
name = "Harry”
start = name[0]
last = name[4]
```

Exercise: initials.py

Given two persons' names, print their Initials as the example below:

- John
- Margie

- → print to video : J&M

Methods: More clear in future lessons

- In python, an object is a software entity that represents a value with certain behavior.
  - The value can be simple, such as a string, or complex, or data file.
- The behavior of an object is given through its **methods**.
  - A method is a collection of programming instructions to carry out a specific task – similar to a function
- But unlike a **function**, which is a standalone operation, a **method** can only be applied to an object of the type for which it was defined.
  - Methods are specific to a type of object
  - Functions are general and can accept arguments of different types
- You can apply the upper() method to any string, like this:

```
  name = "John Smith"
uppercaseName = name.upper() # Sets uppercaseName to "JOHN SMITH"
```

Some Useful String Methods

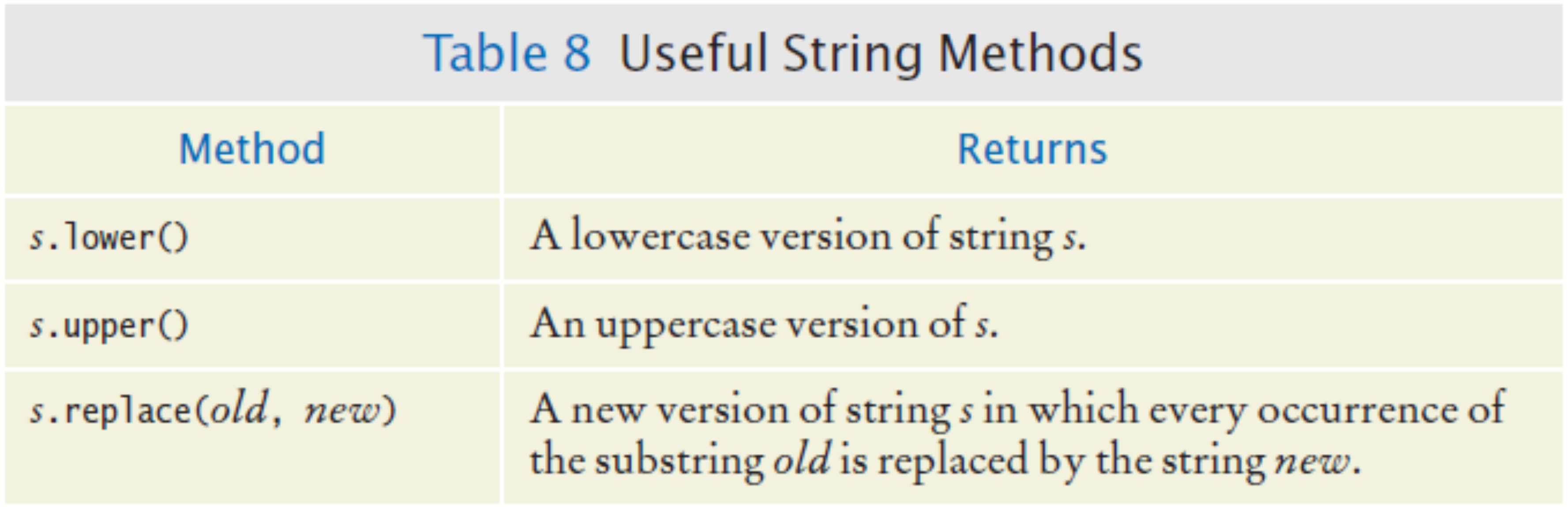


Table 8 Useful String Methods

Method	Returns
`s.lower()`	A lowercase version of string *s*.
`s.upper()`	An uppercase version of *s*.
`s.replace(`*old*`,` *new*`)`	A new version of string *s* in which every occurrence of the substring *old* is replaced by the string *new*.

String Escape Sequences

- How would you print a double quote?
  - Preface the " with a “\” inside the double quoted String

```
print("He said \"Hello\"")
```

- OK, then how do you print a backslash?
  - Preface the \ with another \

```
System.out.print("“C:\\Temp\\Secret.txt“")
```

- Special characters inside Strings
  - Output a newline with a ‘\n’

```
print("*\n**\n***\n")
```

```
*
**
***
```

Exsercise:

- Given a string print the same string in uppercase (string_upper.py)
- Given an arbitrary sentence containing the word «old», substitute this word with the word «new» (string_replace.py)

Input and Output

Input and Output

- You can read a String from the console with the input() function:

```
name = input("Please enter your name")
```

- Converting a String variable to a number can be used if numeric (rather than string input) is needed

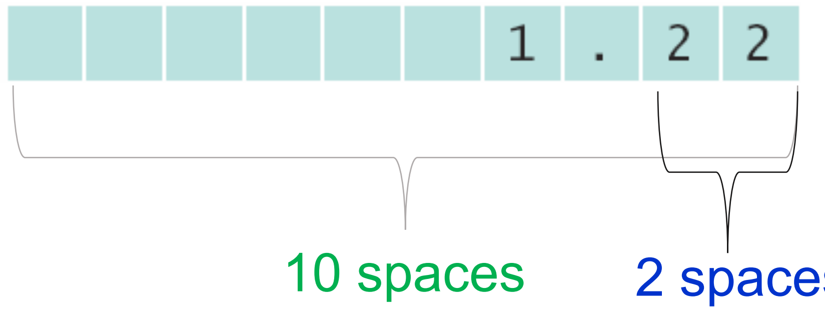
```
age = int(input("Please enter age: "))
```

- The above is equivalent to doing it two steps (getting the input and then converting it to a number):

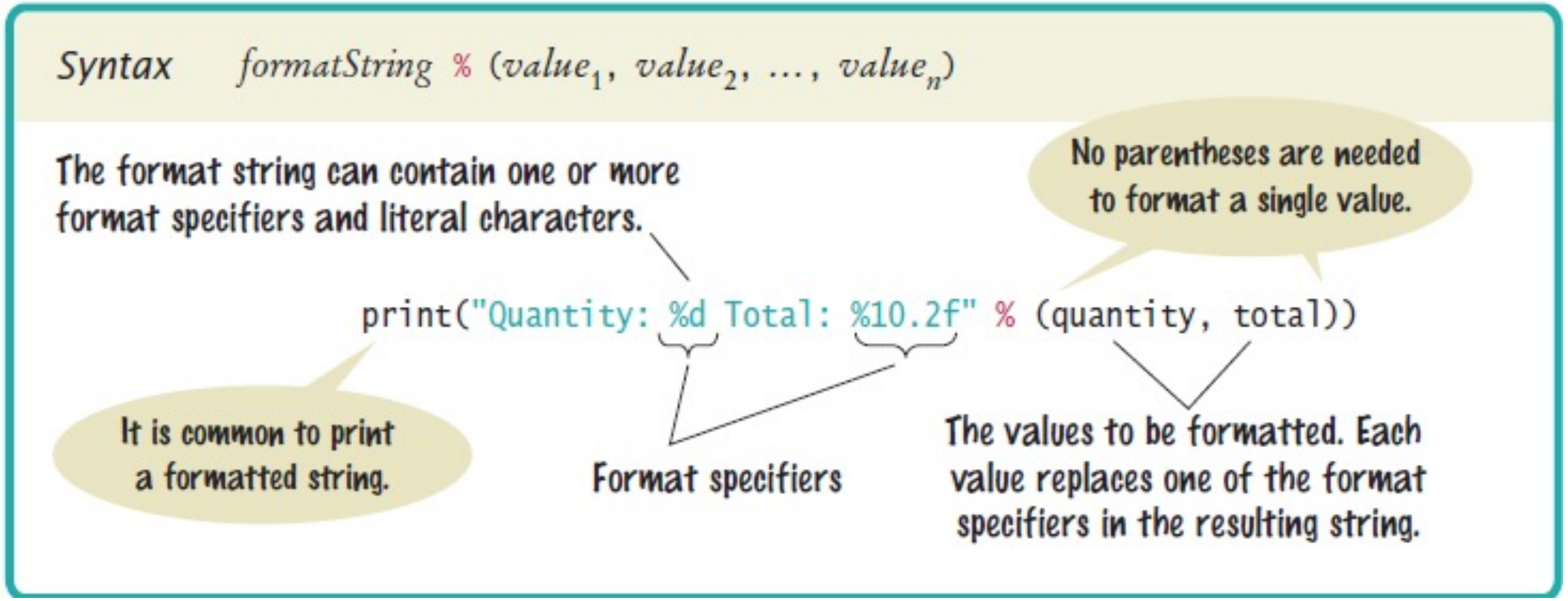
```
aString = input("Please enter age: ")        # String input
age = int(aString)                         # Converted int
```

Formatted output

- Outputting floating point values can look strange:
  ```
  Price per liter:  1.21997
  ```
- To control the output appearance of numeric variables, use formatted output tools such as:
  ```
  print("Price per liter %.2f"  %(price))
  Price per liter: 1.22
  print("Price per liter %10.2f"  %(price))
  Price per liter:      1.22
  ```
- The %10.2f is called a format specifier



Syntax: formatting strings



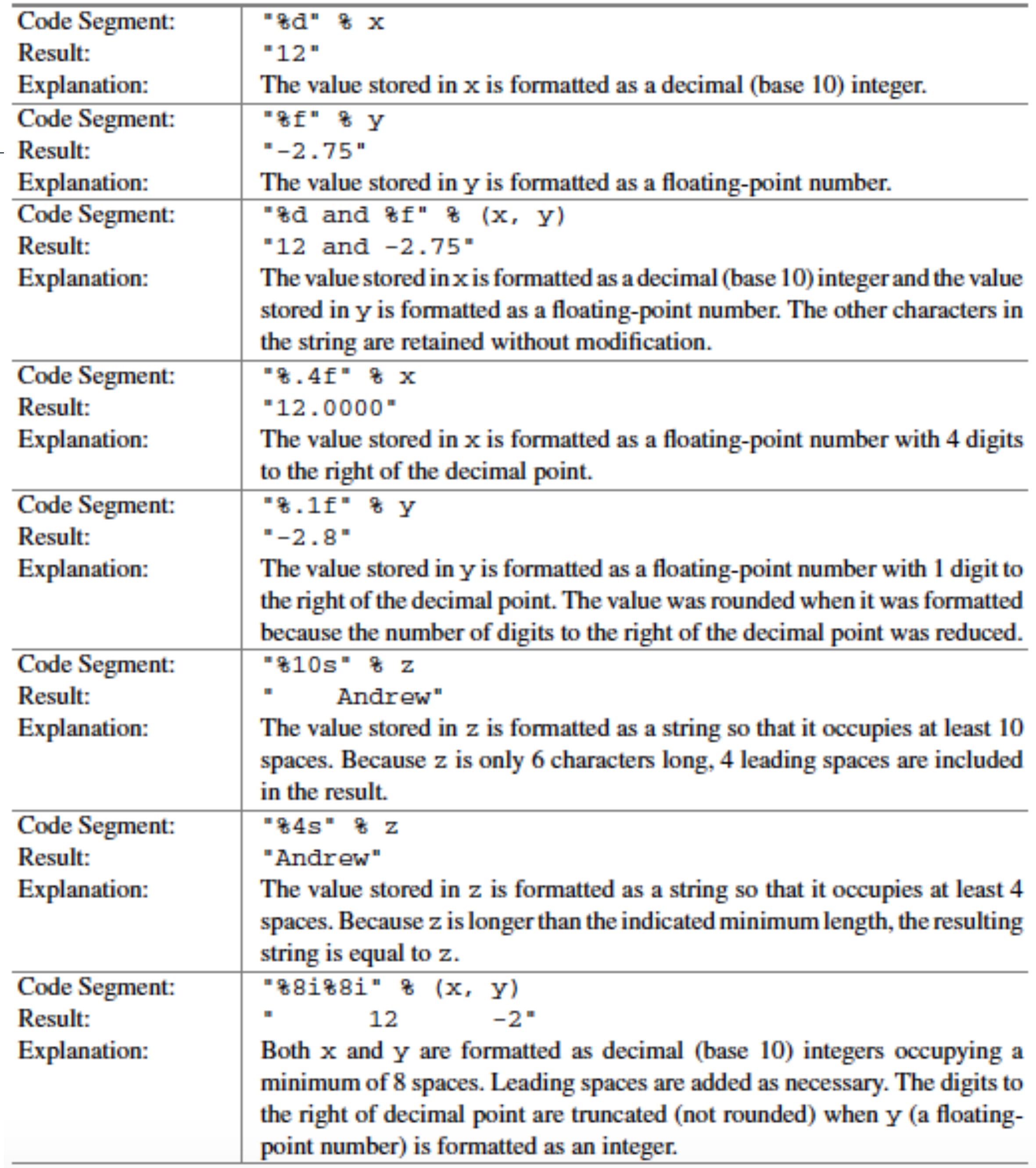
Syntax *formatString* % (*value*$_1$, *value*$_2$, ..., *value*$_n$)

```
print("Quantity: %d Total: %10.2f" % (quantity, total))
```

The format string can contain one or more format specifiers and literal characters.
No parentheses are needed to format a single value.
It is common to print a formatted string.
Format specifiers
The values to be formatted. Each value replaces one of the format specifiers in the resulting string.

Formatting Values

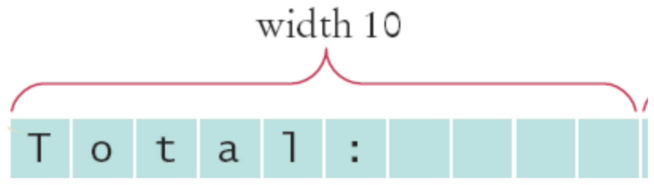
- %f indicates that a value should be formatted as a floating-point number
- %d as a decimal (base-10) integer
- %s as a string
- once the correct formatting characters have been identified, a percent sign (%) is prepended to them.

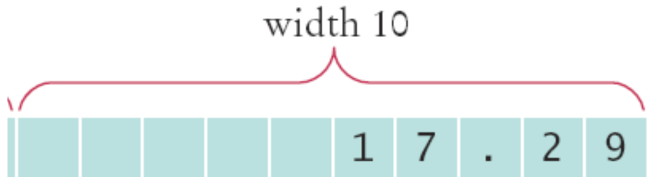


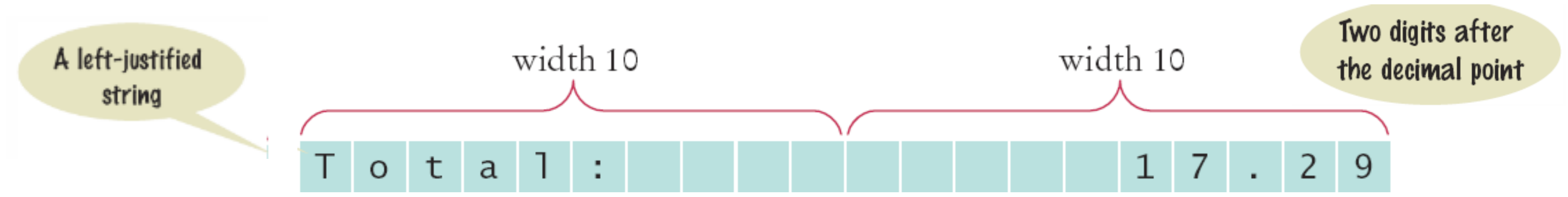
Code Segment:	`"%d" % x`
Result:	`"12"`
Explanation:	The value stored in `x` is formatted as a decimal (base 10) integer.
Code Segment:	`"%f" % y`
Result:	`"-2.75"`
Explanation:	The value stored in `y` is formatted as a floating-point number.
Code Segment:	`"%d and %f" % (x, y)`
Result:	`"12 and -2.75"`
Explanation:	The value stored in `x` is formatted as a decimal (base 10) integer and the value stored in `y` is formatted as a floating-point number. The other characters in the string are retained without modification.
Code Segment:	`"%.4f" % x`
Result:	`"12.0000"`
Explanation:	The value stored in `x` is formatted as a floating-point number with 4 digits to the right of the decimal point.
Code Segment:	`"%.1f" % y`
Result:	`"-2.8"`
Explanation:	The value stored in `y` is formatted as a floating-point number with 1 digit to the right of the decimal point. The value was rounded when it was formatted because the number of digits to the right of the decimal point was reduced.
Code Segment:	`"%10s" % z`
Result:	`"    Andrew"`
Explanation:	The value stored in `z` is formatted as a string so that it occupies at least 10 spaces. Because `z` is only 6 characters long, 4 leading spaces are included in the result.
Code Segment:	`"%4s" % z`
Result:	`"Andrew"`
Explanation:	The value stored in `z` is formatted as a string so that it occupies at least 4 spaces. Because `z` is longer than the indicated minimum length, the resulting string is equal to `z`.
Code Segment:	`"%8i%8i" % (x, y)`
Result:	`"      12      -2"`
Explanation:	Both `x` and `y` are formatted as decimal (base 10) integers occupying a minimum of 8 spaces. Leading spaces are added as necessary. The digits to the right of decimal point are truncated (not rounded) when `y` (a floating-point number) is formatted as an integer.

Format flag examples

- Left Justify a String:
- print("%-10s" %("Total:"))
- Right justify a number with two decimal places
  - print("%10.2f" %(price))
- And you can print multiple values:
  - print("%-10s%10.2f" %("Total: ", price))

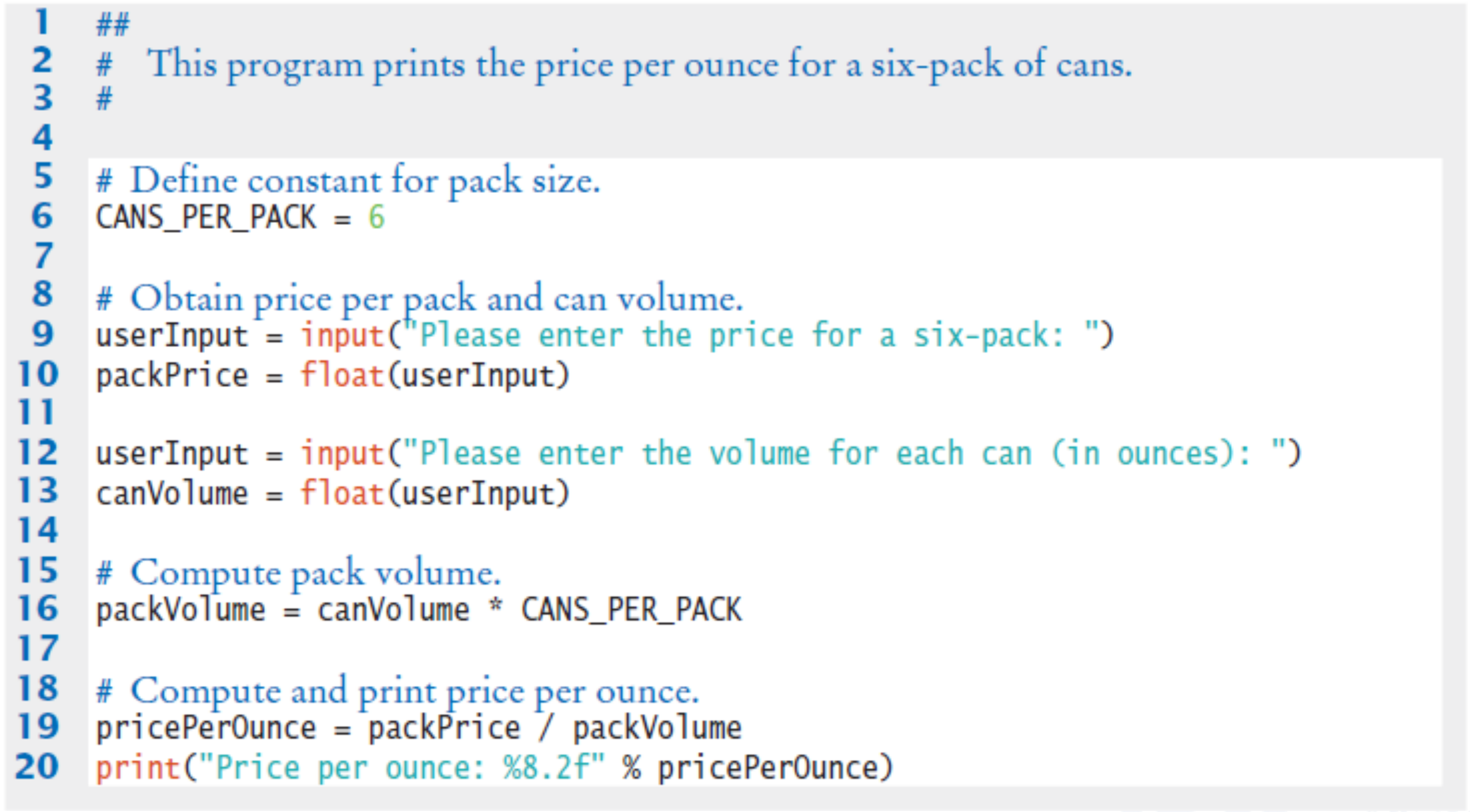






Exemple: Volume2.py

ch02/volume2.py



```
##
#  This program prints the price per ounce for a six-pack of cans.
#

# Define constant for pack size.
CANS_PER_PACK = 6

# Obtain price per pack and can volume.
userInput = input("Please enter the price for a six-pack: ")
packPrice = float(userInput)

userInput = input("Please enter the volume for each can (in ounces): ")
canVolume = float(userInput)

# Compute pack volume.
packVolume = canVolume * CANS_PER_PACK

# Compute and print price per ounce.
pricePerOunce = packPrice / packVolume
print("Price per ounce: %8.2f" % pricePerOunce)
```

Format Specifier Examples: formatting.py

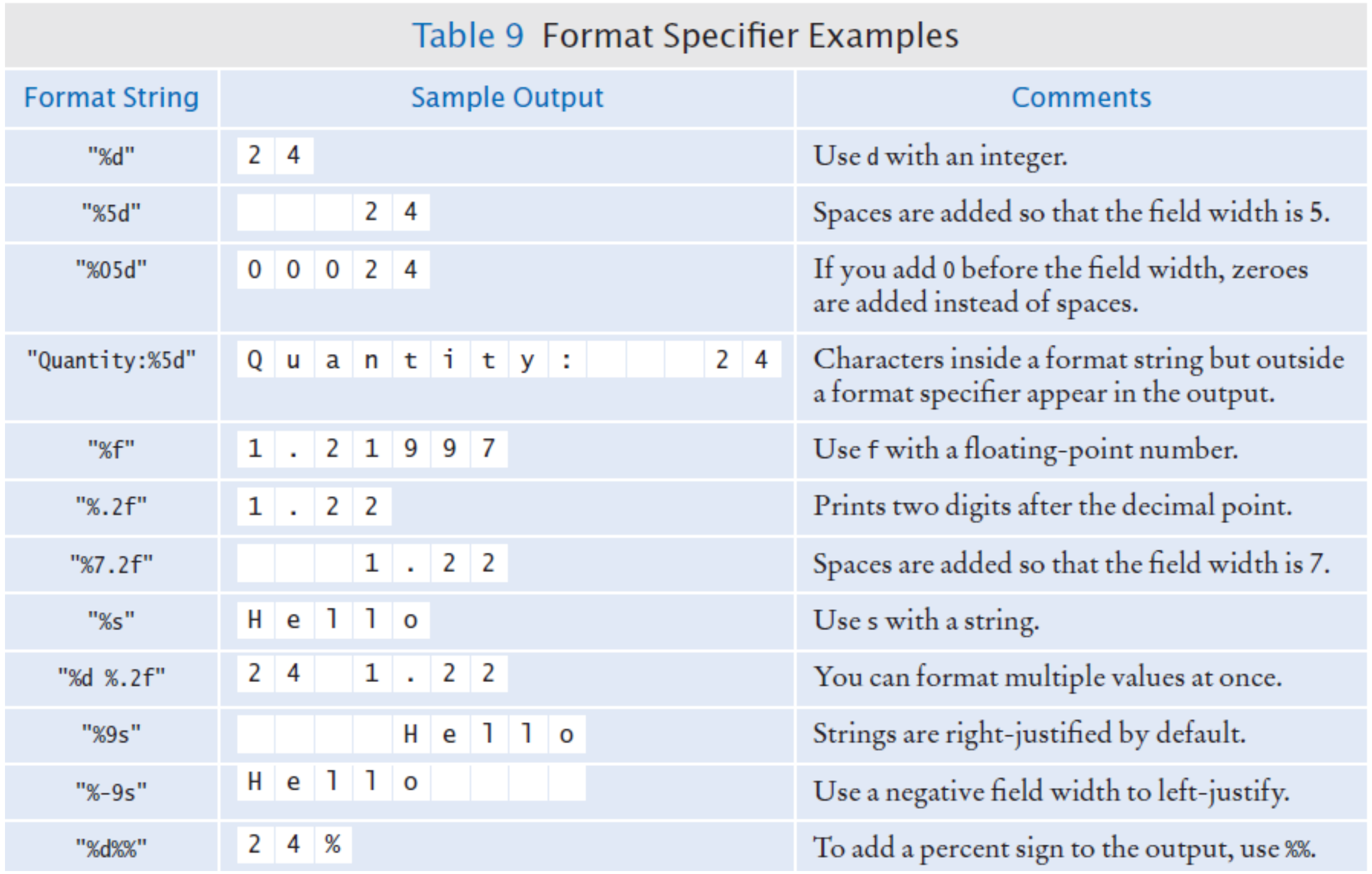


Table 9 Format Specifier Examples

Format String	Sample Output	Comments
"%d"	`24`	Use d with an integer.
"%5d"	`   24`	Spaces are added so that the field width is 5.
"%05d"	`00024`	If you add 0 before the field width, zeroes are added instead of spaces.
"Quantity:%5d"	`Quantity:   24`	Characters inside a format string but outside a format specifier appear in the output.
"%f"	`1.21997`	Use f with a floating-point number.
"%.2f"	`1.22`	Prints two digits after the decimal point.
"%7.2f"	`   1.22`	Spaces are added so that the field width is 7.
"%s"	`Hello`	Use s with a string.
"%d %.2f"	`24 1.22`	You can format multiple values at once.
"%9s"	`    Hello`	Strings are right-justified by default.
"%-9s"	`Hello    `	Use a negative field width to left-justify.
"%d%%"	`24%`	To add a percent sign to the output, use %%.

Example: formatting.py

- Test different formatting styles

Summary: variables

- A variable is a storage location with a name.
- When defining a variable, you must specify an initial value.
- By convention, variable names should start with a lower case letter.
- An assignment statement stores a new value in a variable, replacing the previously stored value.

Summary: operators

- The assignment operator = does not denote mathematical equality.
- Variables whose initial value should not change are typically capitalized by convention.
- The / operator performs a division yielding a value that may have a fractional value.
- The // operator performs a division, the remainder is discarded.
- The % operator computes the remainder of a floor division.

Summary: python overview

- The Python library declares many mathematical functions, such as sqrt() and abs()
- You can convert between integers, floats and strings using the respective functions: int(), float(), str()
- Python libraries are grouped into modules. Use the import statement to use methods from a module.
- Use the input() function to read keyboard input in a console window.

Summary: Strings

- Strings are sequences of characters.
- The len() function yields the number of characters in a String.
- Use the + operator to concatenate Strings; that is, to put them together to yield a longer String.
- In order to perform a concatenation, the + operator requires both arguments to be strings. Numbers must be converted to strings using the str() function.
- String index numbers are counted starting with 0.
- Use the [ ] operator to extract the elements of a String.
- Use the format specifiers to specify how values should be formatted.

MASTER MEIM 2021-2022

Thank you for your attention