PGR107
Python Programming

Lecture 2 – Programming with Numbers and Strings

Kristiania University College

Chapter 2

Programming With Numbers and Strings

Chapter Goals

- To define and use 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

Chapter 2

Programming With Numbers and Strings

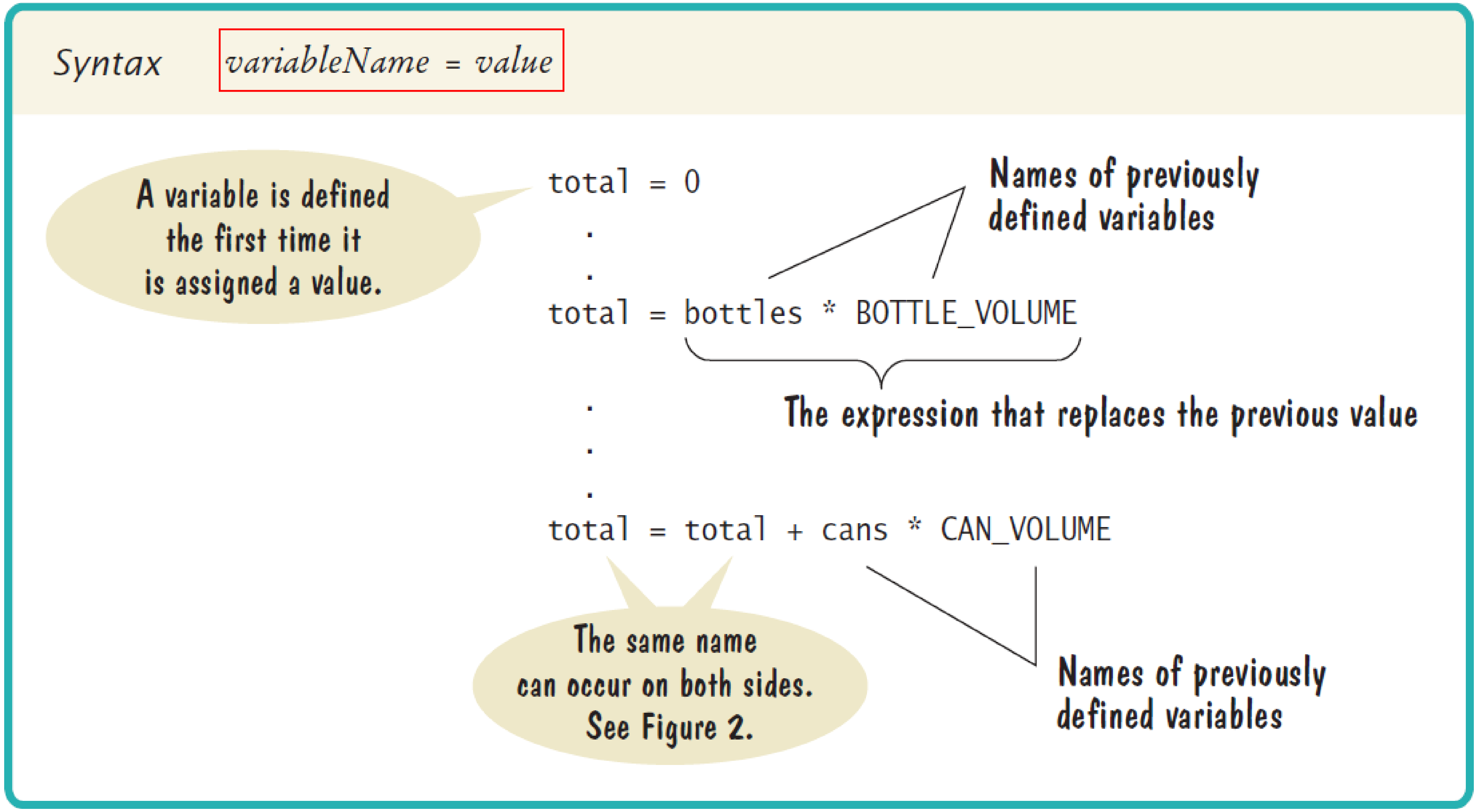
Chapter Goals

- To create programs that read and process inputs, and display the results
- To learn how to use Python strings

Variables

- A **variable** is a storage location in a computer program. Each variable has a name and holds a value.

Assignment Statement


Syntax
variableName = value
total = 0
total = bottles * BOTTLE_VOLUME
total = total + cans * CAN_VOLUME
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

Data Types

- **Primitive data type**: is a data type provided by the language itself.
  - ✓ Numbers
  - ✓ Strings

- **User-defined data type**: is a data type that can be defined by programmers (will be covered in detail in Chapter 9).

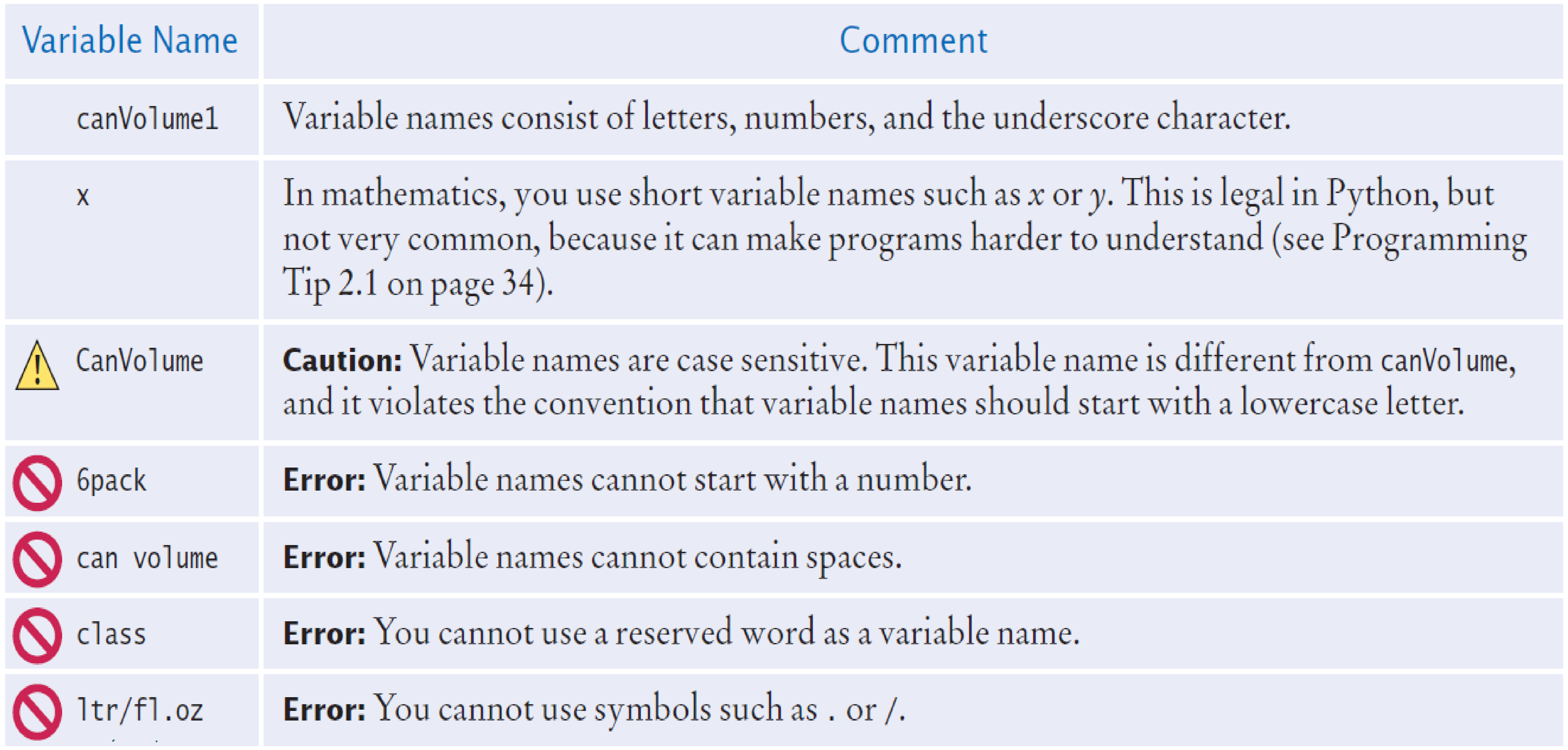
Number Types

- **Integer:** a whole number without a fractional part (int).
- **Floating-point** number: a number with a fractional part (float).
- For example, in Python:
  - a = 5
  - b = 5.5

Numbers 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.

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 34).
⚠ `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 `/`.

Constants

- A constant variable or simply a **constant** is a variable whose value should not be changed after it has been assigned an initial value.

- It is common practice to specify a constant variable with the use of **all capital letters** for its name.
    - ✓ BOTTLE_VOLUME = 2.0
    - ✓ MAX_SIZE = 100

- Example:
    - ✓ totalVolume = bottles * 2
    - ✓ totalVolume = bottles * BOTTLE_VOLUME

Arithmetic Operations

- The symbols + - * / for the arithmetic operations are called **operators**. The combination of variables, numbers, operators, and parentheses is called an **expression**. For example, **(a + b) / 2** is an expression.

- Python uses the exponential operator ** to denote the **power operation**. For example, the Python equivalent of the mathematical expression $\mathbf{a}^2$ is **a ** 2**.

- For example, the mathematical expression:

$$b \times \left(1 + \frac{r}{100}\right)^n$$

becomes:

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

Arithmetic Operations

- print (7 / 4) $\longrightarrow$ **1.75**

- print (7 // 4) $\longrightarrow$ **1**
- **floor division** using the **//** operator. For positive integers, floor division computes the quotient and discards the fractional part.

- print (7 % 4) $\longrightarrow$ **3**
- **%** is **remainder** operator (or modulus/mod operator).

Operator Precedence

20 - 3 * 4 = ?

( )

**

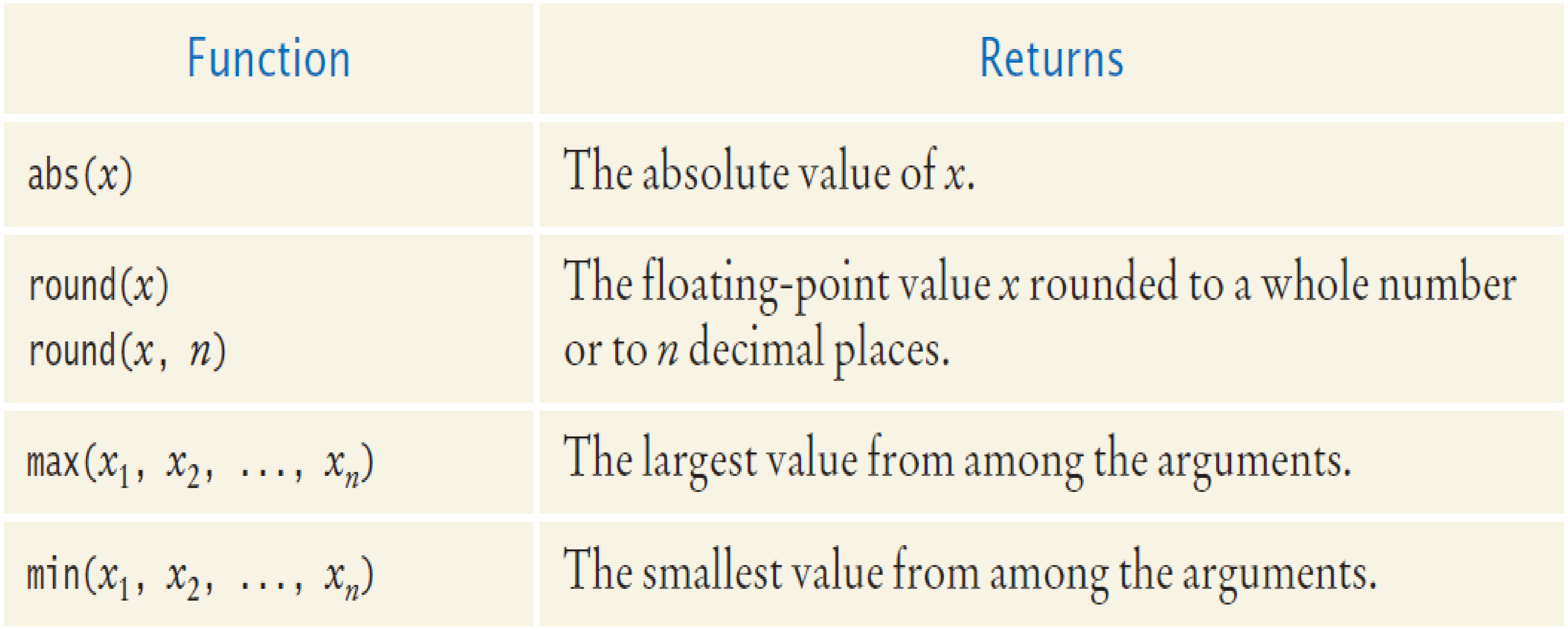
* /

+ -

(10 - 3) + 6 – 2 ** 2 = ?

Calling Functions

- **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, ..., x_n)	The largest value from among the arguments.
min(x_1, x_2, ..., x_n)	The smallest value from among the arguments.

Mathematical Functions

- A **library** is a collection of code that has been written and translated by someone else, ready for you to use in your program.

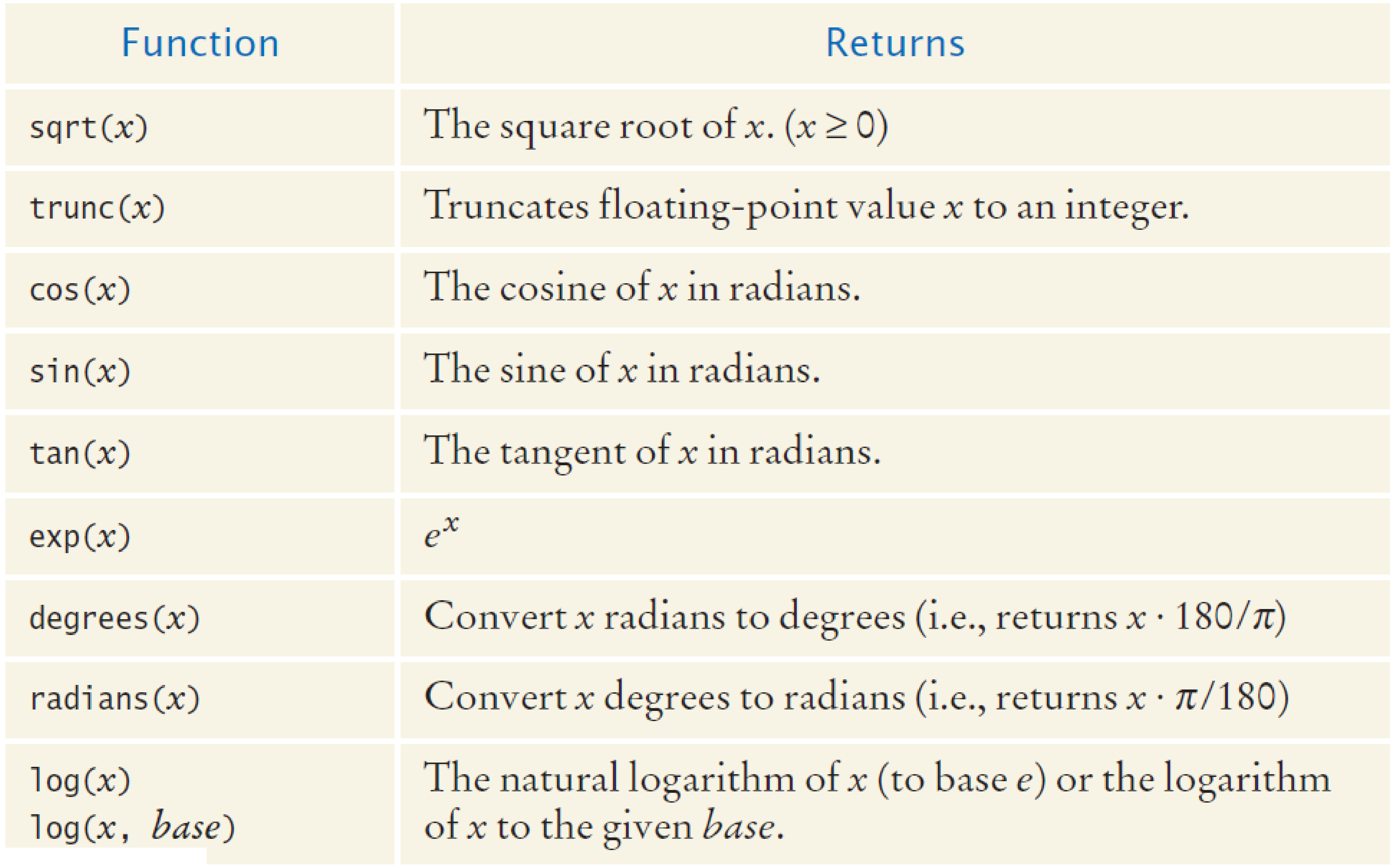
- A **standard library** is a library that is considered part of the language and must be included with any Python system.

- Python’s standard library is organized into **modules**.
- math Module:

```
from math import sqrt
y = sqrt(x)
```

Mathematical Functions

- **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*.

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.

Strings

- A **string** is a sequence of characters. For example, the string "**Hello**" is a sequence of five characters.
- print (“Hello”)   OR
- greeting = “Hello” $\longrightarrow$ print (greeting)

Computes the length of a string

- length = **len** (“Hello”)
- print (length)   # length is 5

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

Concatenation

- In Python, + operator is used to concatenate two strings.

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

- print (name) # HarryMorgan

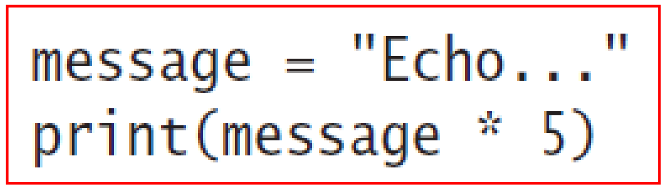
```
name = firstName + " " + lastName
```

- print (name) # Harry Morgan

Repetition

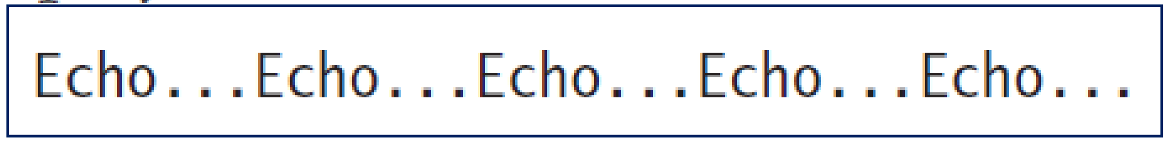
- A string of any length can be repeated using the * operator. For example:

- dashes = “-”
- print (dashes * 50)



```
message = "Echo..."
print(message * 5)
```

display



```
Echo...Echo...Echo...Echo...Echo...
```

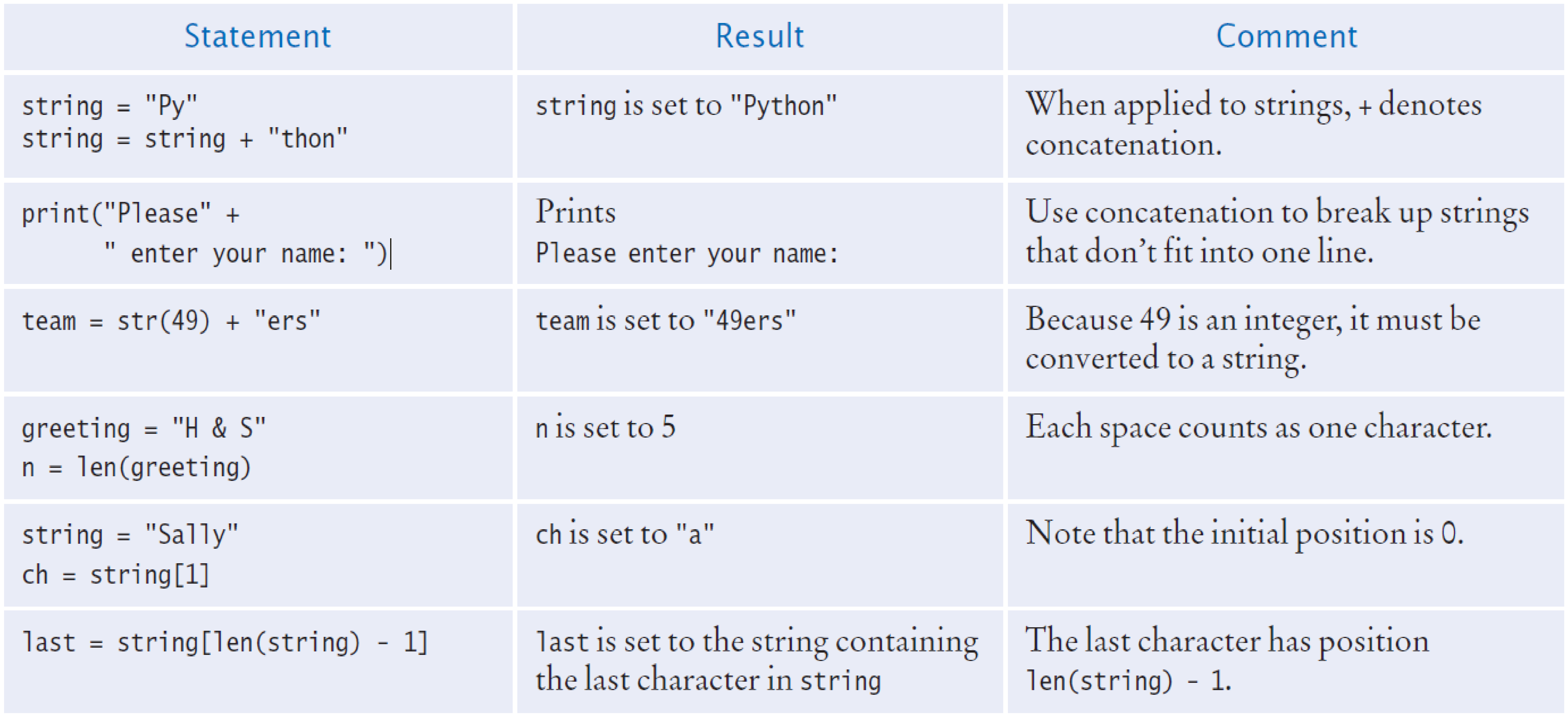
Converting Between Numbers and Strings

- Sometimes it is necessary to convert a numerical value to a string. For example, suppose you need to append a number to the end of a string. You cannot concatenate a string and a number:
- name = “Agent ” + 1234   # Can only concatenate strings

- name = “Agent ” + **str** (1234)

- To turn a string containing a number into a numerical value:
- id = **int** (“1234”)
- price = **float** (“12.34”)

Strings 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.

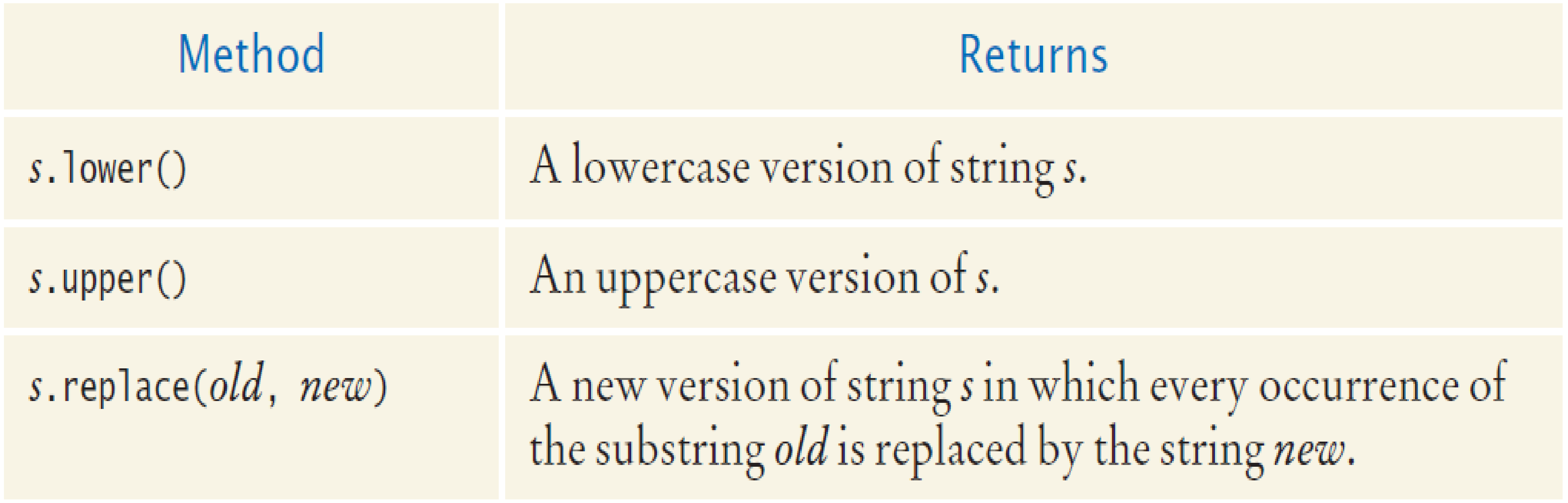
`team = str(49) + "ers"`	`team` is set to `"49ers"`	Because 49 is an integer, it must be converted to a string.

`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`.

Useful Strings 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*.

Useful Strings Methods

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

```
print(name.lower())    # Prints john smith
```

```
name2 = name.replace("John", "Jane")    # Sets name2 to "Jane Smith"
```

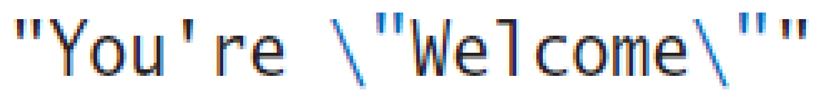
Character Values

- Python provides two functions related to character encodings. The **ord** function returns the number used to represent a given character. The **chr** function returns the character associated with a given code. For example,

- print (ord (“H”))   # 72
- print (ord (“a”))   # 97
- print (chr (97))   # a
- print (chr (72))   # H

Escape Sequence

- Sometimes you may need to include both single and double quotes in a literal string. For example, to include double quotes around the word Welcome in the literal string “You’re Welcome”, precede the quotation marks with a backslash (\), like this:



```
"You're \"Welcome\""
```

- Another common escape sequence is **\n**, which denotes a **newline character**.

- print (“Welcome**\n**to**\n**MEK1300”)

User Input

- You can make your programs more flexible if you ask the program user for inputs rather than using fixed values.

- When a program asks for user input, it should first print a message that tells the user which input is expected. Such a message is called a **prompt**.

```
first = input("Enter your first name: ")
```

Use the input function to read keyboard input.

User Input

- **Exercise:** Write a program to obtain a first name and a last name from the user, and prints the pair of initials.

- **Sample Run**:

```
Enter your first name: John
Enter your last name: Smith
Your initials: JS
```

Numerical Input

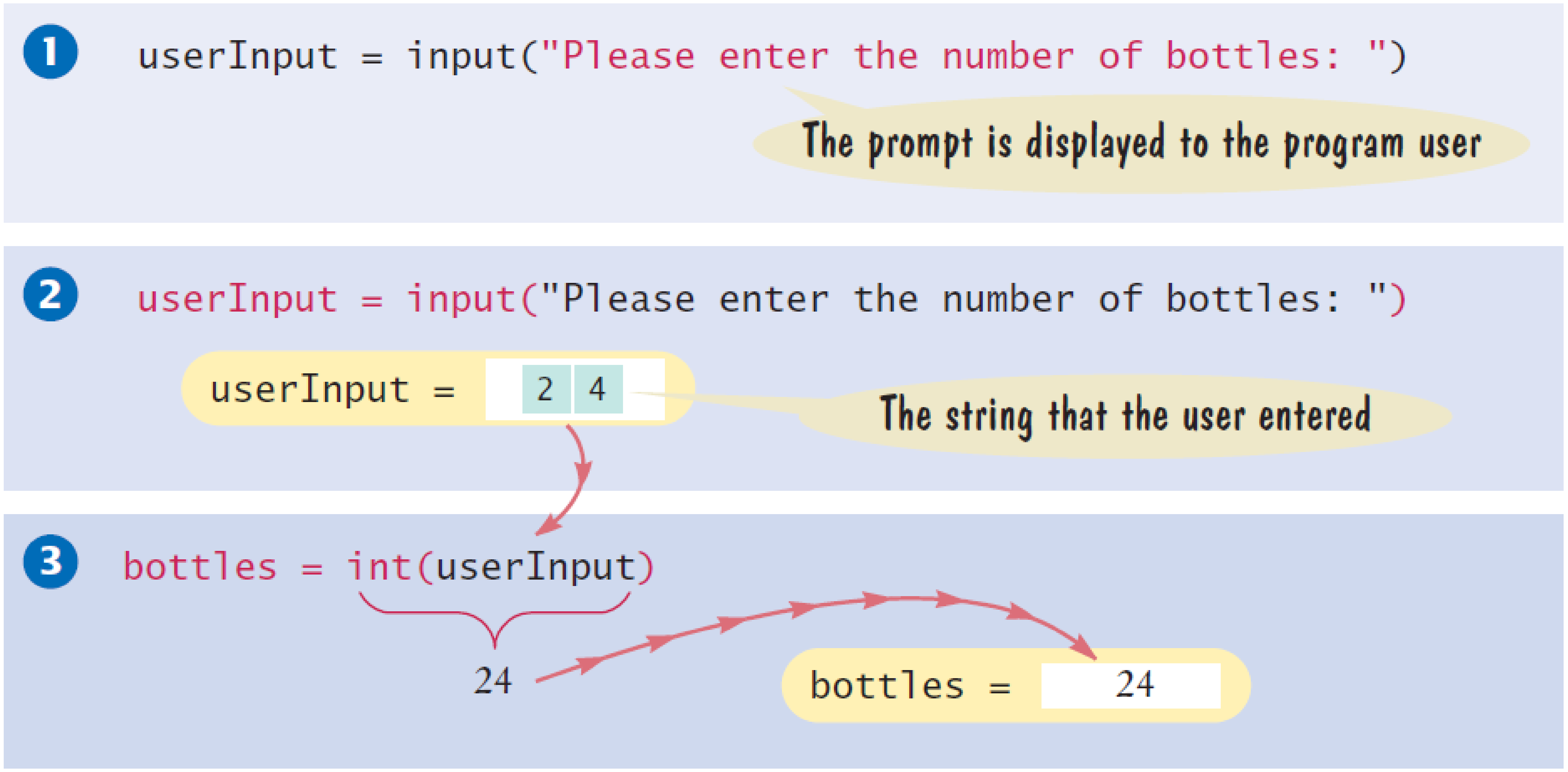
- The **input** function can only obtain a **string** of text from the user. But what if we need to obtain a numerical value?

```
userInput = input("Please enter the number of bottles: ")
bottles = int(userInput)
```

```
userInput = input("Enter price per bottle: ")
price = float(userInput)
```

To read an integer or floating-point value, use the `input` function followed by the `int` or `float` function.

Extracting an Integer Value


1
userInput = input("Please enter the number of bottles: ")
The prompt is displayed to the program user
2
userInput = input("Please enter the number of bottles: ")
userInput = 2 4
The string that the user entered
3
bottles = int(userInput)
24
bottles = 24

Formatted Output

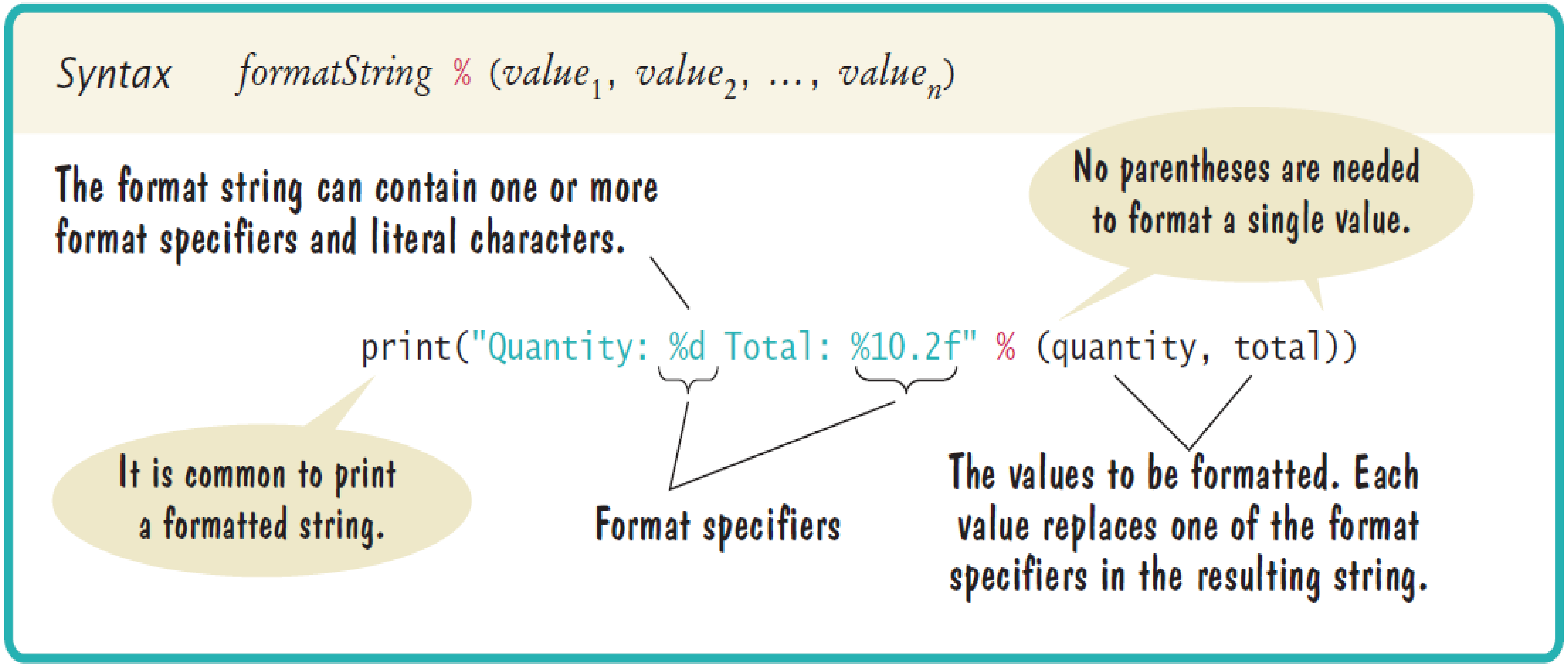
- When you print the result of a computation, you often want to control its appearance. For example, you want the output to look like:

Price per liter: 1.22

instead of

Price per liter: 1.215962441314554

Formatted Output



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

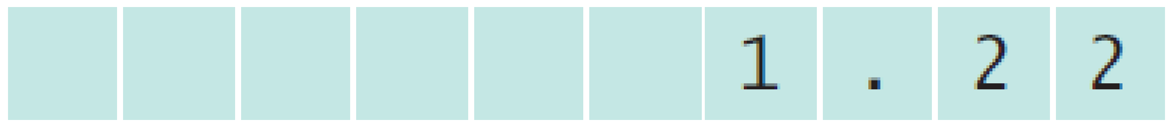
```
print("Quantity: %d Total: %10.2f" % (quantity, total))
```

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.

Formatted Output

- **Example 1:**
- price = 1.215962441314554

```
print ( “%10.2f” % price)
```

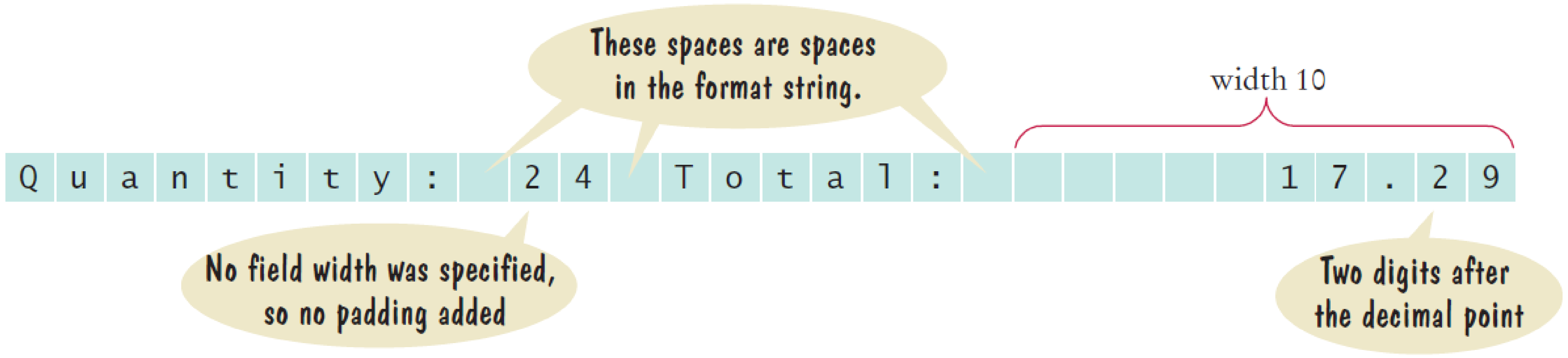


```
      1.22
```

Formatted Output

- **Example 2:**
- quantity = 24
- total = 17.289

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


These spaces are spaces in the format string.
width 10
Q u a n t i t y : 2 4 T o t a l : 1 7 . 2 9
No field width was specified, so no padding added
Two digits after the decimal point

Formatted Output

- **Example 3:**

```python
title1 = "Quantity:"
title2 = "Price:"
print("%10s %10d" % (title1, 24))
print("%10s %10.2f" % (title2, 17.29))
```

```
 Quantity:         24
    Price:      17.29
```

- **Example 4:**

```python
print("%-10s %10d" % (title1, 24))
print("%-10s %10.2f" % (title2, 17.29))
```

```
Quantity:          24
Price:          17.29
```

Format Specifier: More 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 %%.

Formatted String

```
name = “John”
age = 35

print (“Hi ” + name + “. You are ” + str (age) + “ years old.”)

print (“Hi {}. You are {} years old.”.format (name, age))

print (f ”Hi {name}. You are {age} years old.”)
```

End of Chapter 2