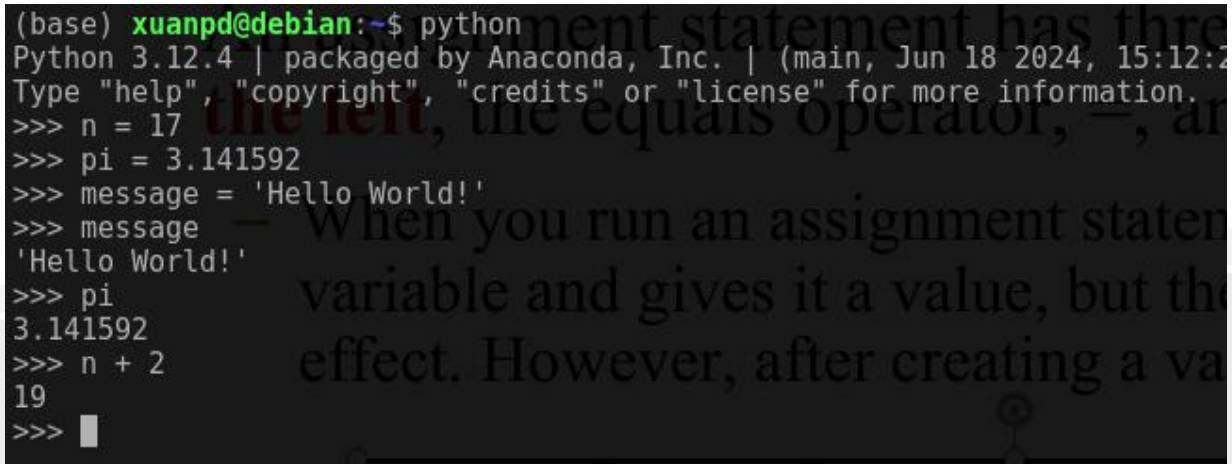
Chapter 2
Variables and Statements

A. Downey, *Think Python: How to Think Like a Computer Science,* 3rd ed., O'Reilly, 2024.

Contents

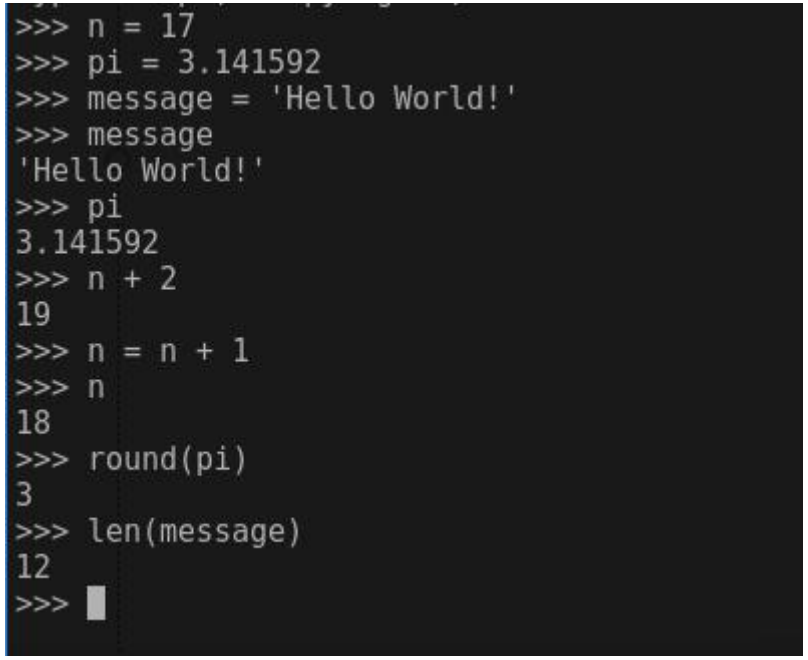
1. Variables

- A variable is a name that refers to a value. To create a variable, we can write an assignment statement.
- An assignment statement has three parts: the name of the **variable on the left**, the equals operator, =, and an expression on the right.
  - When you run an assignment statement, there is no output. Python creates the variable and gives it a value, but the assignment statement has no visible effect. However, after creating a variable, you can use it as an expression.



1. Variables

- You can also use a variable as part of an expression with arithmetic operators or you can use a variable when you call a function.

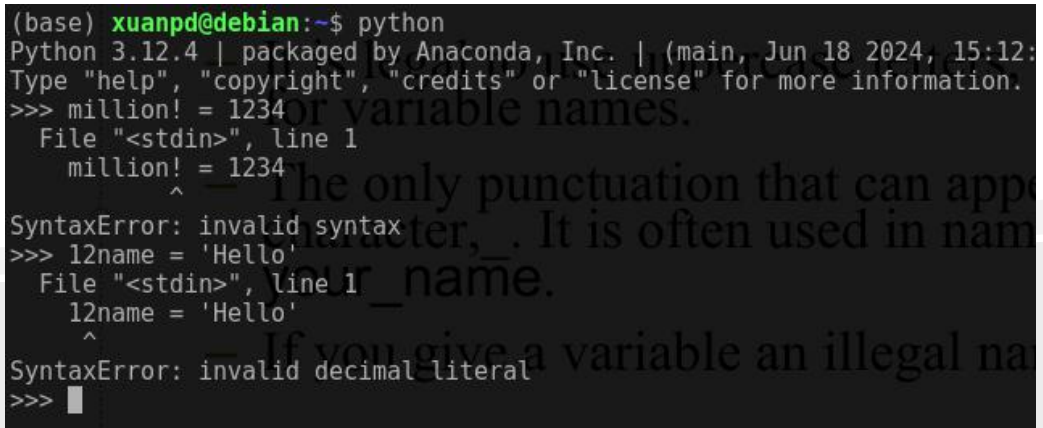


2. State Diagrams

- A common way to represent variables on paper is to write the name with an arrow pointing to its value:
  - n → 2
  - pi → 3.141592
  - message → ‘Hello World!’
- This kind of figure is called a state diagram because it shows what state each of the variables is in (think of it as the variable’s state of mind).

3. Variable Names

- Variable names can be as long as you like.
- They can contain both letters and numbers, but they can't begin with a number.
  - It is legal to use uppercase letters, but it is conventional to use only lowercase for variable names.
  - The only punctuation that can appear in a variable name is the underscore character, _. It is often used in names with multiple words, such as your_name.
  - If you give a variable an illegal name, you get a syntax error.



3. Variable Names

- Keywords can’t be used as variable names.
  - Here’s a complete list of Python’s keywords:

```
False      await      else       import     pass
None       break      except     in         raise
True       class      finally    is         return
and        continue   for        lambda     try
as         def        from       nonlocal   while
assert     del        global     not        with
async      elif       if         or         yield
```

4. The import Statement

- In order to use some Python modules, you have to **import** them. For example, the following statement imports the math module:

  import math

- A **module** is a collection of variables and functions. The math module provides a variable called pi that contains the value of the mathematical constant denoted π .
  - To use a variable, function, ... in a module, you have to use the dot operator (.) between the name of the module and the name of the variable, function, ...

```
(base) xuanpd@debian:~$ python
Python 3.12.4 | packaged by Anaconda, Inc. | (main, Jun 18 2024, 15:12:
Type "help", "copyright", "credits" or "license" for more information.
>>> import math
>>> math.pi
3.141592653589793
>>> math.sqrt(16)
4.0
>>> math.pow(4, 2)
16.0
>>> 4 ** 2
16
>>>
```

5. Expressions and Statements

- An expression can be a single value, like an integer, floating-point number, or string. It can also be a collection of values and operators. And it can include variable names and function calls.
- A statement is a unit of code that has an effect, but **no value**.
  - An import statement has an effect—it imports a module so we can use the values and functions it contains—but it has no visible effect

```
>>> import math
>>> n = 20
>>> n + 30 + round(math.pi) * 2
56
>>>
```

- Computing the value of an expression is called **evaluation**. Running a statement is called **execution**.

6. The print Function

- To display a value, you can use the print function.
  - You can also use the print function to display a sequence of values separated by commas.
    - Notice that the print function puts a space between the values.

```
chapter2.py > ...
1   n = 17
2   pi = 3.141592653589793
3   message = 'And now for something completely different'
4
5   print(message)
6   print(n, pi, message)
```

```
And now for something completely different
17 3.141592653589793 And now for something completely different
```

7. Arguments

- When you call a function, the expression in parentheses is called an **argument**.
- Some of the functions we've seen so far take only one argument, like int.

Some functions take two, three ... arguments.

  - >>> int('12') -----> 12
  - >>> math.pow(3, 2) -----> 9

- Some can take additional arguments that are **optional**.
  - For example, int can take a second argument that specifies the base of the number:
    - >>> int('11', 2) -----> 3
  - round also takes an optional second argument, which is the number of decimal places to round off to:
    - >>> round(math.pi, 3) -----> 3.142

7. Arguments

- Some functions can take any number of arguments, like print:
  - >>> print('first', 'second', 'third', 'fourth')
- If you call a function and provide too many arguments, that's a TypeError:
  - >>> float('123.45', 2)

  TypeError: float expected at most 1 argument, got 2
- If you provide too few arguments, that's also a TypeError:
  - >>> math.pow(3)

  TypeError: pow expected 2 arguments, got 1
- And if you provide an argument with a type the function can't handle, that's a TypeError, too:
  - >>> math.sqrt('25')

  TypeError: must be real number, not str

8. Comments

- As programs get bigger and more complicated, they get more difficult to read.
- Formal languages are dense, and it is often difficult to look at a piece of code and figure out what it is doing and why.
- For this reason, it is a good idea to add notes to your programs to explain in natural language what the program is doing. These notes are called **comments**, and they start with the # symbol.
  - Everything from the # to the end of the line is ignored—it has no effect on the execution of the program.
  - Comments are most useful when they document non-obvious features of the code. It is reasonable to assume that the reader can figure out *what* the code does; it is more useful to explain *why*.

8. Comments

chapter2.py > ...

```
1
2   # number of seconds in 42:42
3   seconds = 42 * 60 + 42
4
5   print(seconds)
6
7   miles = 10 / 1.61  # 10 kilometers in miles
8   print(miles)
```

```
2562
6.211180124223602
```