7. Selection

7.1. Boolean Values and Boolean Expressions

7.2. Logical operators

7.3. Precedence of Operators

7.4. Conditional Execution: Binary Selection

7.5. Omitting the else Clause: Unary Selection

7.6. Nested conditionals

7.7. Chained conditionals

7.8. Boolean Functions

7.8.1. More Unit Testing

7.9. Glossary

7.10. Exercises

7.1. Boolean Values and Boolean Expressions

The Python type for storing true and false values is called bool, named after the British mathematician, George Boole. George Boole created Boolean Algebra, which is the basis of all modern computer arithmetic.

There are only two boolean values. They are True and False. Capitalization is important, since true and false are not boolean values (remember Python is case sensitive).

1 print(True)

2 print(type(True))

3 print(type(False))

4

​

Note

Boolean values are not strings!

It is extremely important to realize that True and False are not strings. They are not surrounded by quotes. They are the only two values in the data type bool. Take a close look at the types shown below.

1 print(type(True))

2 print(type("True"))

3

A boolean expression is an expression that evaluates to a boolean value. The equality operator, ==, compares two values and produces a boolean value related to whether the two values are equal to one another.

1 print(5 == 5)

2 print(5 == 6)

3

​

In the first statement, the two operands are equal, so the expression evaluates to True. In the second statement, 5 is not equal to 6, so we get False.

The == operator is one of six common comparison operators; the others are:

x != y # x is not equal to y

x > y # x is greater than y

x < y # x is less than y

x >= y # x is greater than or equal to y

x <= y # x is less than or equal to y

Although these operations are probably familiar to you, the Python symbols are different from the mathematical symbols. A common error is to use a single equal sign (=) instead of a double equal sign (==). Remember that = is an assignment operator and == is a comparison operator. Also, there is no such thing as =< or =>.

Note too that an equality test is symmetric, but assignment is not. For example, if a == 7 then 7 == a. But in Python, the statement a = 7 is legal and 7 = a is not. (Can you explain why?)

Check your understanding

select-1-5: Which of the following is a Boolean expression? Select all that apply.

A. True

B. 3 == 4

C. 3 + 4

D. 3 + 4 == 7

E. "False"

7.2. Logical operators

There are three logical operators: and, or, and not. The semantics (meaning) of these operators is similar to their meaning in English. For example, x > 0 and x < 10 is true only if x is greater than 0 and at the same time, x is less than 10. How would you describe this in words? You would say that x is between 0 and 10, not including the endpoints.

n % 2 == 0 or n % 3 == 0 is true if either of the conditions is true, that is, if the number is divisible by 2 or divisible by 3. In this case, one, or the other, or both of the parts has to be true for the result to be true.

Finally, the not operator negates a boolean expression, so not x > y is true if x > y is false, that is, if x is less than or equal to y.

1 x = 5

2 print(x > 0 and x < 10)

3

​4 n = 25

5 print(n % 2 == 0 or n % 3 == 0)

6

Common Mistake!

There is a very common mistake that occurs when programmers try to write boolean expressions. For example, what if we have a variable number and we want to check to see if its value is 5,6, or 7. In words we might say: “number equal to 5 or 6 or 7”. However, if we translate this into Python, number == 5 or 6 or 7, it will not be correct. The or operator must join the results of three equality checks. The correct way to write this is number == 5 or number == 6 or number == 7. This may seem like a lot of typing but it is absolutely necessary. You cannot take a shortcut.

Check your understanding

select-2-2: What is a correct Python expression for checking to see if a number stored in a variable x is between 0 and 5?

A. x > 0 and < 5

B. x > 0 or x < 5

C. x > 0 and x < 5

7.3. Precedence of Operators

We have now added a number of additional operators to those we learned in the previous chapters. It is important to understand how these operators relate to the others with respect to operator precedence. Python will always evaluate the arithmetic operators first (\*\* is highest, then multiplication/division, then addition/subtraction). Next comes the relational operators. Finally, the logical operators are done last. This means that the expression x\*5 >= 10 and y-6 <= 20 will be evaluated so as to first perform the arithmetic and then check the relationships. The and will be done last. Although many programmers might place parentheses around the two relational expressions, it is not necessary.

The following table summarizes the precedence discussed so far from highest to lowest. See Operator precedence table for all the operators introduced in this book.

Level

Category

Operators

7(high)

exponent

\*\*

6

multiplication

\*,/,//,%

5

addition

+,-

4

relational

==,!=,<=,>=,>,<

3

logical

not

2

logical

and

1(low)

logical

Check your understanding

select-3-2: Which of the following properly expresses the precedence of operators (using parentheses) in the following expression: 5\*3 > 10 and 4+6==11

A. ((5\*3) > 10) and ((4+6) == 11)

B. (5\*(3 > 10)) and (4 + (6 == 11))

C. ((((5\*3) > 10) and 4)+6) == 11

D. ((5\*3) > (10 and (4+6))) == 11

Here is an animation for the above expression:

5 \* 3 > 10 and 4 + 6 == 11

5 \* 3 > 10 and 4 + 6 == 11

7.4. Conditional Execution: Binary Selection

In order to write useful programs, we almost always need the ability to check conditions and change the behavior of the program accordingly. Selection statements, sometimes also referred to as conditional statements, give us this ability. The simplest form of selection is the if statement. This is sometimes referred to as binary selection since there are two possible paths of execution.

1

x = 15

2

​3 if x % 2 == 0:

4 print(x, "is even")

5 else:

6 print(x, "is odd")

7

The syntax for an if statement looks like this:

if BOOLEAN EXPRESSION:

STATEMENTS\_1 # executed if condition evaluates to True

else:

STATEMENTS\_2 # executed if condition evaluates to False

The boolean expression after the if statement is called the condition. If it is true, then the immediately following indented statements get executed. If not, then the statements indented under the else clause get executed.

Flowchart of a if statement with an else

../\_images/flowchart\_if\_else.png

As with the function definition from the last chapter and other compound statements like for, the if statement consists of a header line and a body. The header line begins with the keyword if followed by a boolean expression and ends with a colon (:).

The more indented statements that follow are called a block.

Each of the statements inside the first block of statements is executed in order if the boolean expression evaluates to True. The entire first block of statements is skipped if the boolean expression evaluates to False, and instead all the statements under the else clause are executed.

There is no limit on the number of statements that can appear under the two clauses of an if statement, but there has to be at least one statement in each block.

Each compound statement includes a heading and all the following further-indented statements in the block after the heading. The if - else statement is an unusual compound statement because it has more than one part at the same level of indentation as the if heading, (the else clause, with its own indented block).

Check your understanding

select-4-3: How many statements can appear in each block (the if and the else) in a conditional statement?

A. Just one.

B. Zero or more.

C. One or more.

D. One or more, and each must contain the same number.

select-4-4: What does the following code print (choose from output a, b, c or nothing)?

if 4 + 5 == 10:

print("TRUE")

else:

print("FALSE")

A. TRUE

B. FALSE

C. TRUE on one line and FALSE on the next

D. Nothing will be printed

select-4-5: What does the following code print?

if 4 + 5 == 10:

print("TRUE")

else:

print("FALSE")

print("TRUE")

a. TRUE

b.

TRUE

FALSE

c.

FALSE

TRUE

d.

TRUE

FALSE

TRUE

A. Output a

B. Output b

C. Output c

D. Output d

7.5. Omitting the else Clause: Unary Selection

Another form of the if statement is one in which the else clause is omitted entirely. This creates what is sometimes called unary selection. In this case, when the condition evaluates to True, the statements are executed. Otherwise the flow of execution continues to the statement after the body of the if.

1 x = 10

2 if x < 0:

3 print("The negative number ", x, " is not valid here.")

4 print("This is always printed")

5

What would be printed if the value of x is negative? Try it.

Check your understanding

select-5-3: What does the following code print?

x = -10

if x < 0:

print("The negative number ", x, " is not valid here.")

print("This is always printed")

a.

This is always printed

b.

The negative number -10 is not valid here

C. This is always printed

The negative number -10 is not valid here

A. Output a

B. Output b

C. Output c

D. It will cause an error because every if must have an else clause.

select-5-4: Will the following code cause an error?

x = -10

if x < 0:

print("The negative number ", x, " is not valid here.")

else:

print(x, " is a positive number")

else:

print("This is always printed")

A. No

B. Yes

7.6. Nested conditionals

One conditional can also be nested within another. For example, assume we have two integer variables, x and y. The following pattern of selection shows how we might decide how they are related to each other.

if x < y:

print("x is less than y")

else:

if x > y:

print("x is greater than y")

else:

print("x and y must be equal")

The outer conditional contains two branches. The second branch (the else from the outer) contains another if statement, which has two branches of its own. Those two branches could contain conditional statements as well.

Here is a complete program that defines values for x and y. Run the program and see the result. Then change the values of the variables to change the flow of control.

1 x = 10

2 y = 10

3

​4 if x < y:

5 print("x is less than y")

6 else:

7 if x > y:

8 print("x is greater than y")

9 else:

10 print("x and y must be equal")

11

Note

In some programming languages, matching the if and the else is a problem. However, in Python this is not the case. The indentation pattern tells us exactly which else belongs to which if.

Check your understanding

select-6-3: Will the following code cause an error?

x = -10

if x < 0:

print("The negative number ", x, " is not valid here.")

else:

if x > 0:

print(x, " is a positive number")

else:

print(x," is 0")

A. No

B. Yes

7.7. Chained conditionals

Python provides an alternative way to write nested selection such as the one shown in the previous section. This is sometimes referred to as a chained conditional.

if x < y:

print("x is less than y")

elif x > y:

print("x is greater than y")

else:

print("x and y must be equal")

elif is an abbreviation of else if. Again, exactly one branch will be executed. There is no limit of the number of elif statements but only a single (and optional) final else statement is allowed and it must be the last branch in the statement.

Each condition is checked in order. If the first is false, the next is checked, and so on. If one of them is true, the corresponding branch executes, and the statement ends. Even if more than one condition is true, only the first true branch executes.

Here is the same program using elif.

1 x = 10

2 y = 10

3

​4 if x < y:

5 print("x is less than y")

6 elif x > y:

7 print("x is greater than y")

8 else:

9 print("x and y must be equal")

10

​

Check your understanding

select-7-3: Which of I, II, and III below gives the same result as the following nested if?

# nested if-else statement

x = -10

if x < 0:

print("The negative number ", x, " is not valid here.")

else:

if x > 0:

print(x, " is a positive number")

else:

print(x, " is 0")

I.

if x < 0:

print("The negative number ", x, " is not valid here.")

else x > 0:

print(x, " is a positive number")

else:

print(x, " is 0")

II.

if x < 0:

print("The negative number ", x, " is not valid here.")

elif x > 0:

print(x, " is a positive number")

else:

print(x, " is 0")

III.

if x < 0:

print("The negative number ", x, " is not valid here.")

if x > 0:

print(x, " is a positive number")

else:

print(x, " is 0")

A. I only

B. II only

C. III only

D. II and III

E. I, II, and III

select-7-4: What will the following code print if x = 3, y = 5, and z = 2?

if x < y and x < z:

print("a")

elif y < x and y < z:

print("b")

else:

print("c")

A. a

B. b

C. c

SavvyPython

View:

Textbook

Social

Search

User

Scratch Activecode

Help

This Chapter

7.8. Boolean Functions

We have already seen that boolean values result from the evaluation of boolean expressions. Since the result of any expression evaluation can be returned by a function (using the return statement), functions can return boolean values. This turns out to be a very convenient way to hide the details of complicated tests. For example:

1

def isDivisible(x, y):

2

if x % y == 0:

3

result = True

4

else:

5

result = False

6

​

7

return result

8

​

9

print(isDivisible(10, 5))

10

​

Activity: 7.8.1 ActiveCode (ch06\_boolfun1)

The name of this function is isDivisible. It is common to give boolean functions names that sound like yes/no questions. isDivisible returns either True or False to indicate whether the x is or is not divisible by y.

We can make the function more concise by taking advantage of the fact that the condition of the if statement is itself a boolean expression. We can return it directly, avoiding the if statement altogether:

def isDivisible(x, y):

return x % y == 0

Boolean functions are often used in conditional statements:

if isDivisible(x, y):

... # do something ...

else:

... # do something else ...

It might be tempting to write something like if isDivisible(x, y) == True: but the extra comparison is redundant. You only need an == expression if you are comparing some other type than boolean. (isDivisible(x, y) == False can also be made more concise as not isDivisible(x, y)). The following example shows the isDivisible function at work. Notice how descriptive the code is when we move the testing details into a boolean function. Try it with a few other actual parameters to see what is printed.

1

def isDivisible(x, y):

2

return x % y == 0

3

​

4

if isDivisible(10, 5):

5

print("That works")

6

else:

7

print("Those values are no good")

8

​

Activity: 7.8.2 ActiveCode (ch06\_boolfun2)

Here is the same program in codelens. When we evaluate the if statement in the main part of the program, the evaluation of the boolean expression causes a call to the isDivisible function. This is very easy to see in codelens.

1 def isDivisible(x, y):

2 return x % y == 0

3

4 if isDivisible(10, 5):

5 print("That works")

6 else:

7 print("Those values are no good")

line that just executed

next line to execute

Check your understanding

select-8-4: What is a Boolean function?

A. A function that returns True or False

B. A function that takes True or False as an argument

C. The same as a Boolean expression

select-8-5: Is the following statement legal in a Python function (assuming x, y and z are defined to be numbers)?

return x + y < z

A. Yes

B. No

7.8.1. More Unit Testing

When we write unit tests, we should also consider output equivalence classes that result in significantly different results.

The isDivisible function can return either True or False. These two different outputs give us two equivalence classes. We then choose inputs that should give each of the different results. It is important to have at least one test for each output equivalence class.

1 def isDivisible(x, y):

2 '''is x evenly divisible by y?'''

3 return x % y == 0

4

​5 if \_\_name\_\_ == "\_\_main\_\_":

6 import test

7

​

Extend the program …

Starting on line 7, write two unit tests (that should pass), one for each output equivalence class.

7.9. Glossary

block

A group of consecutive statements with the same indentation.

body

The block of statements in a compound statement that follows the header.

boolean expression

An expression that is either true or false.

boolean function

A function that returns a boolean value. The only possible values of the bool type are False and True.

boolean value

There are exactly two boolean values: True and False. Boolean values result when a boolean expression is evaluated by the Python interepreter. They have type bool.

branch

One of the possible paths of the flow of execution determined by conditional execution.

chained conditional

A conditional branch with more than two possible flows of execution. In Python chained conditionals are written with if ... elif ... else statements.

comparison operator

One of the operators that compares two values: ==, !=, >, <, >=, and <=.

condition

The boolean expression in a conditional statement that determines which branch is executed.

conditional statement

A statement that controls the flow of execution depending on some condition. In Python the keywords if, elif, and else are used for conditional statements.

logical operator

One of the operators that combines boolean expressions: and, or, and not.

modulus operator

An operator, denoted with a percent sign ( %), that works on integers and yields the remainder when one number is divided by another.

nesting

One program structure within another, such as a conditional statement inside a branch of another conditional statement.

7.10. Exercises

What do these expressions evaluate to?

3 == 3

3 != 3

3 >= 4

not (3 < 4)

Give the logical opposites of these conditions. You are not allowed to use the not operator.

a > b

a >= b

a >= 18 and day == 3

a >= 18 or day != 3

Write a function which is given an exam mark, and it returns a string — the grade for that mark — according to this scheme:

Mark

Grade

>= 90

A

[80-90)

B

[70-80)

C

[60-70)

D

< 60

F

The square and round brackets denote closed and open intervals. A closed interval includes the number, and open interval excludes it. So 79.99999 gets grade C , but 80 gets grade B.

Test your function by printing the mark and the grade for a number of different marks.

Modify the turtle bar chart program from the previous chapter so that the bar for any value of 200 or more is filled with red, values between [100 and 200) are filled yellow, and bars representing values less than 100 are filled green.

In the turtle bar chart program, what do you expect to happen if one or more of the data values in the list is negative? Go back and try it out. Change the program so that when it prints the text value for the negative bars, it puts the text above the base of the bar (on the 0 axis).

Write a function findHypot. The function will be given the length of two sides of a right-angled triangle and it should return the length of the hypotenuse. (Hint: x \*\* 0.5 will return the square root, or use sqrt from the math module)

Write a function called is\_even(n) that takes an integer as an argument and returns True if the argument is an even number and False if it is odd.

Now write the function is\_odd(n) that returns True when n is odd and False otherwise.

Modify is\_odd so that it uses a call to is\_even to determine if its argument is an odd integer.

Write a function is\_rightangled which, given the length of three sides of a triangle, will determine whether the triangle is right-angled. Assume that the third argument to the function is always the longest side. It will return True if the triangle is right-angled, or False otherwise.

Hint: floating point arithmetic is not always exactly accurate, so it is not safe to test floating point numbers for equality. If a good programmer wants to know whether x is equal or close enough to y, they would probably code it up as

if abs(x - y) < 0.001: # if x is approximately equal to y

Extend the above program so that the sides can be given to the function in any order.

3 criteria must be taken into account to identify leap years:

The year is evenly divisible by 4;

If the year can be evenly divided by 100, it is NOT a leap year, unless;

The year is also evenly divisible by 400. Then it is a leap year.

Write a function that takes a year as a parameter and returns True if the year is a leap year, False otherwise.

Implement the calculator for the date of Easter.

The following algorithm computes the date for Easter Sunday for any year between 1900 to 2099.

Ask the user to enter a year. Compute the following:

a = year % 19

b = year % 4

c = year % 7

d = (19 \* a + 24) % 30

e = (2 \* b + 4 \* c + 6 \* d + 5) % 7

dateofeaster = 22 + d + e

Special note: The algorithm can give a date in April. Also, if the year is one of four special years (1954, 1981, 2049, or 2076) then subtract 7 from the date.

Your program should print an error message if the user provides a date that is out of range.