Booleans and Conditionals

Announcement

- Office Hour on Wednesday morning: 9 am – 11 am
- Zoom ID: 638 358 6495
- Quiz next Tuesday

Learning Goals

- Use logical operators on Booleans to compute whether an expression is True or False
- Use conditionals when reading and writing algorithms that make choices based on data
- Use nesting of control structures to create complex control flow
- Debug logical errors by using the scientific method

Logical Operators

Booleans are values that can be True or False

In week 1, we learned about the Boolean type, which can be one of two values: True or False.

Until now, we've made Boolean values by comparing different values, such as:

```
x < 5
s == "Hello"
7 >= 2
```

Logical Operations Combine Booleans

We aren't limited to only evaluating a single Boolean comparison! We can combine Boolean values using logical operations. We'll learn about three – and, or, and not.

Combining Boolean values will let us check complex requirements while running code.

and Operation Checks Both

The and operation takes two Boolean values and evaluates to True if both values are True. In other words, it evaluates to False if either value is False.

We use and when we want to require that both conditions be met at the same time.

Example:

(x >= 0) and (x < 10)

a	b	a and b
True	True	**True**
True	False	False
False	True	False
False	False	False

or Operation Checks Either

The or operation takes two Boolean values and evaluates to True if either value is True. In other words, it only evaluates to False if both values are False.

We use or when there are multiple valid conditions to choose from.

Example:

```
(day == "Saturday") or (day == "Sunday")
```

a	b	a or b
True	True	True
True	False	True
False	True	True
False	False	**False**

not Operation Reverses Result

Finally, the not operation takes a single Boolean value and switches it to the opposite value (negates it). not True becomes False, and not False becomes True.

We use not to switch the result of a Boolean expression. For example, not

(x < 5) is the same as x >= 5.

Example:

not (x == 0)

a	not a
True	False
False	True

Activity: Guess the Result

If x = 10, what will each of the following expressions evaluate to?

```
x < 25 and x > 15
```

```
x < 25 or x > 15
```

```
not (x > 5 and x < 10)
```

```
(x > 5) or ((x**2 > 50) and (x == 20))
```

```
((x > 5) or (x**2 > 50)) and (x == 20)
```

Conditionals

Conditionals Make Decisions

With Booleans, we can make a new type of code called a conditional. Conditionals are a form of a control structure – they let us change the direction of the code based on the value that we provide.

To write a conditional (`if` statement), we use the following structure:

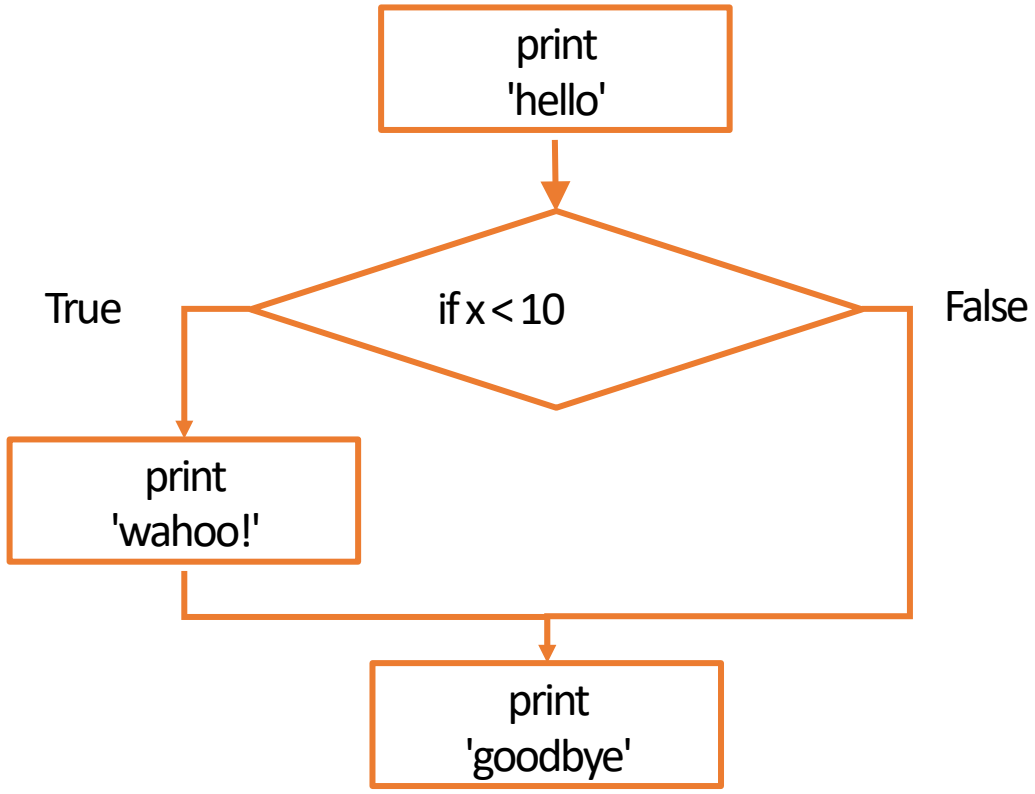
```
if <BooleanExpression>:
    <bodyIfTrue>
```

Note that, like a function definition, the top line of the `if` statement ends with a colon, and the body of the `if` statement is indented.

Flow Charts Show Code Choices

We'll use a flow chart to demonstrate how Python executes an if statement based on the values provided.

```
print("hello")
if x < 10:
        print("wahoo!")
print("goodbye")
```



wahoo! is only printed if x is less than 10. But hello and goodbye are always printed.

The Body of an If Can Have Many Statements

The body of an if statement can have any number of statements in it. As with function definitions, each statement of the body is on a separate line and indented. The body ends when the next line of code is unindented.

```
print("hello")
if x < 10:
        print("wahoo!")
        print("wahoo!")
print("goodbye")
```

if x < 10, prints:

hello
wahoo!
wahoo!
goodbye

if x >= 10, prints:

hello goodbye

Else Clauses Allow Alternatives

Sometimes we want a program to do one of two alternative actions based on the condition. In this case, instead of writing two `if` statements, we can write a single `if` statement and add an `else`.

The `else` is executed when the Boolean expression is `False`.

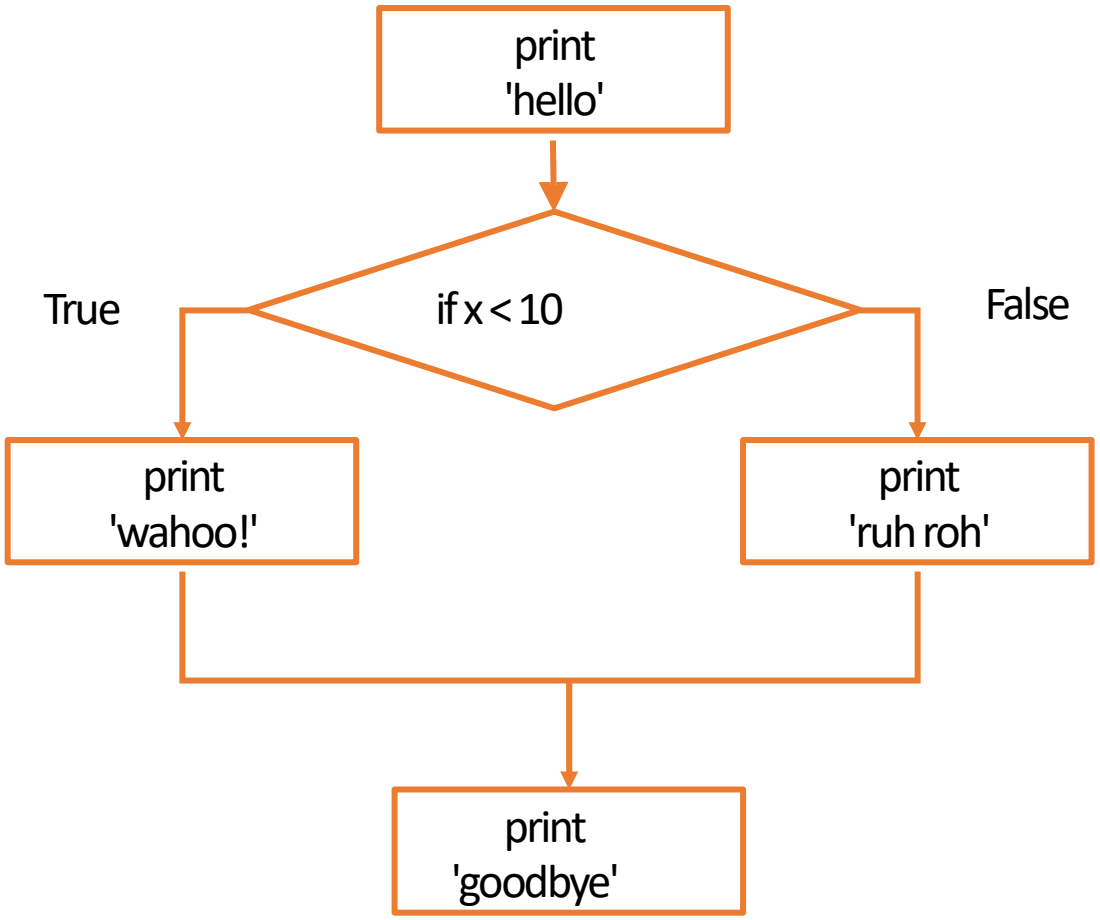
```
if <BooleanExpression>:
        <bodyIfTrue>
else:
        <bodyIfFalse>
```

} if clause

} else clause

Updated Flow Chart Example

```
print("hello")
if x < 10:
        print("wahoo!")
else:
        print("ruh roh")
print("goodbye")
```



Activity: Conditional Prediction

Prediction Exercise: What will the following code print?

```
x = 5
if x > 10:
    print("Up high!")
else:
    print("Down low!")
```

Question: How can we change the program state to print the other string instead?

Question: Can we change the state to make the if/else statement print out both statements?

Else Must Be Paired With If

It's impossible to have an else clause by itself, as it would have no condition to be the alternative to.

Therefore, every else must be paired with an if. On the other hand, every if can have at most one else.

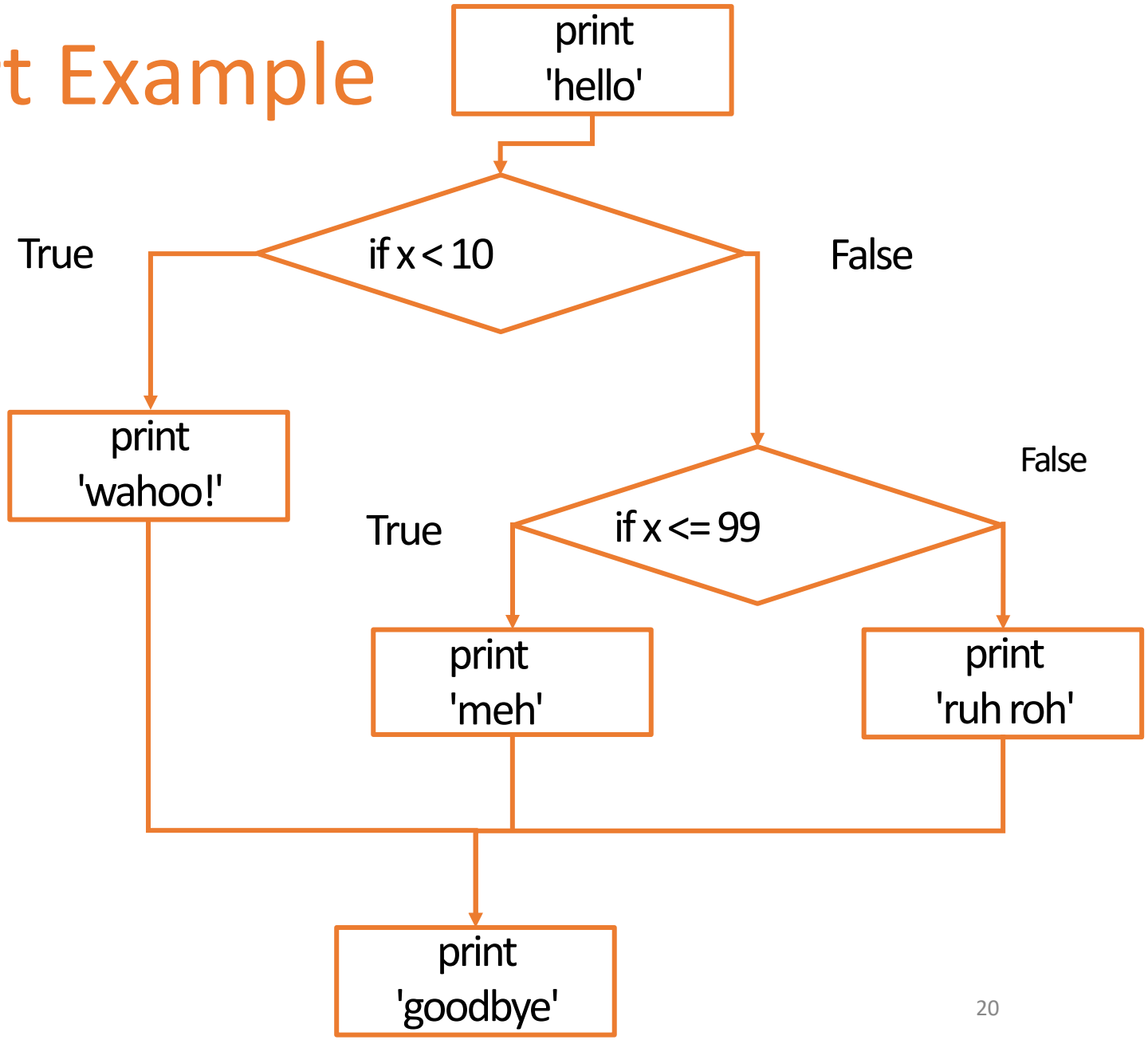
Elif Implements Multiple Alternatives

Finally, we can use `elif` statements to add alternatives with their own conditions to `if` statements. An `elif` is like an `if`, except that it is checked only if all previous conditions evaluate to `False`.

```
if <BooleanExpressionA>:
        <body If A True>
elif <BooleanExpressionB>:
        <body If A False And B True>
else:
        <body If Both False>
```

Updated Flow Chart Example

```
print("hello")
if x < 10:
        print("wahoo!")  elif
x <= 99:
        print("meh")
else:
        print("ruh roh")
print("goodbye")
```



Conditional Statements Join Clauses Together

We can have more than one elif clause associated with an if statement. In fact, we can have as many as we need! But, as with else, an elif must be associated with an if (or a previous elif).

In general, a conditional statement is an if clause with zero or more elifclauses and an optional else clause that are all joined together. These joined clauses can be considered a single control structure. Only one clause will have its body executed.

Example: gradeCalculator

Let's write a few lines of code that takes a grade as a number, then prints the letter grade that corresponds to that number grade.

90+ is an A, 80-90 is a B, 70-80 is a C, 60-70 is a D, and below 60 is an R.

Short-Circuit Evaluation

When Python evaluates a logical expression, it acts lazily. It only evaluates the second part if it needs to. This is called short-circuit evaluation.

When checking `x and y`, if `x` is `False`, the expression can never be `True`. Therefore, Python doesn't even evaluate `y`.

When checking `x or y`, if `x` is `True`, the expression can never be `False`. Python doesn't evaluate `y`.

This is a handy method for keeping errors from happening. For example:

```
if type(x) == type(y) and x < y:
        print("Smaller:", x)
```

Nesting Control Structures

Nesting Creates More Complex Control Flow

Now that we have a control structure, we can put `if` statements inside of `if` statements.

In general, we'll be able to nest control structures inside of other control structures. We can also nest control structures inside of function definitions.

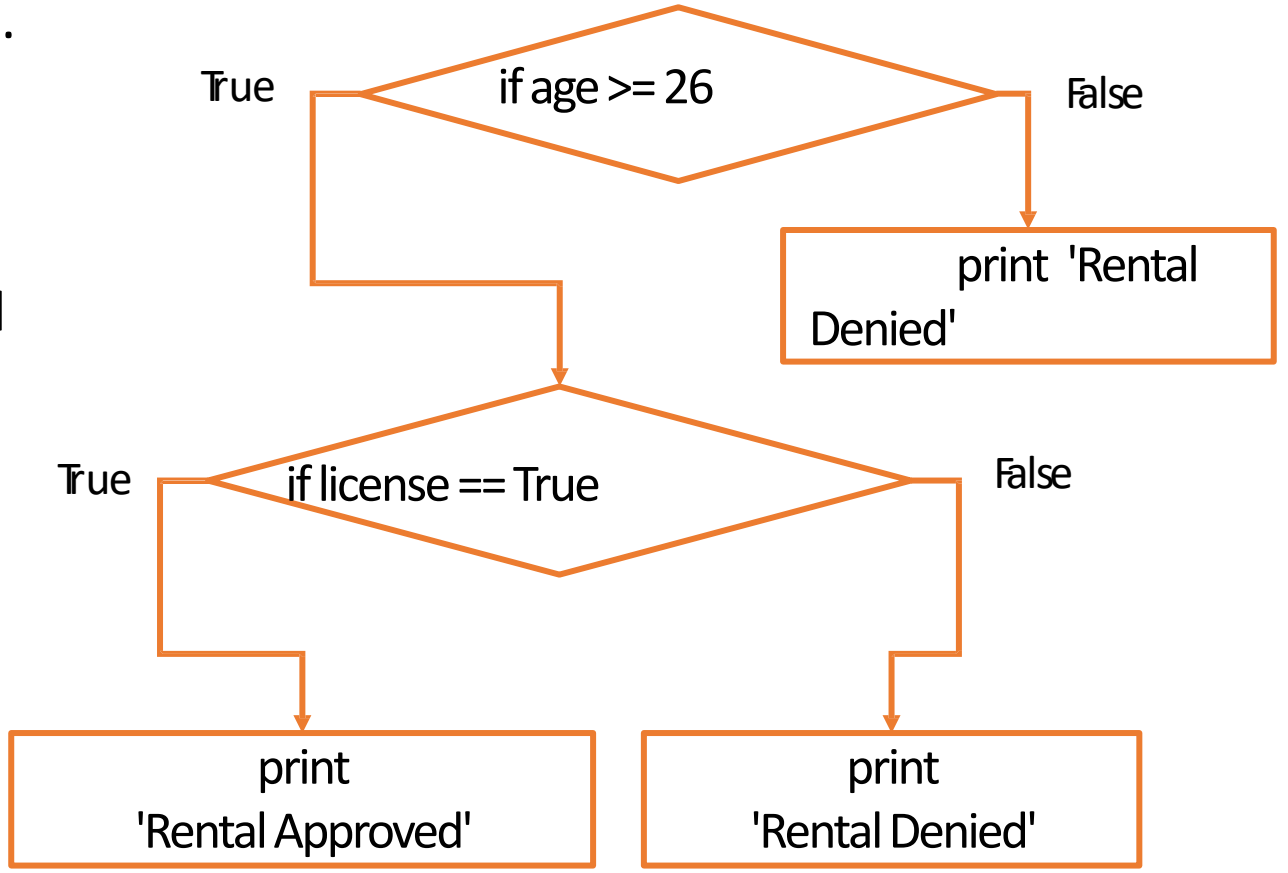
In program syntax, we demonstrate that a control structure is nested by indenting the code so that it's in the outer control structure's body.

Example: Car rental program

Consider code that determines if a person can rent a car based on their age (are they at least 26) and whether they have a driver's license.

We can use one if statement to check their age, then a second (nested inside the first) to check the license. We'll only print 'Rental Approved' if both if conditions evaluate to True.

```
if age >= 26:
        if license == True:
                print("Rental Approved")
        else:
                print("Rental Denied")
else:
        print("Rental Denied")
```

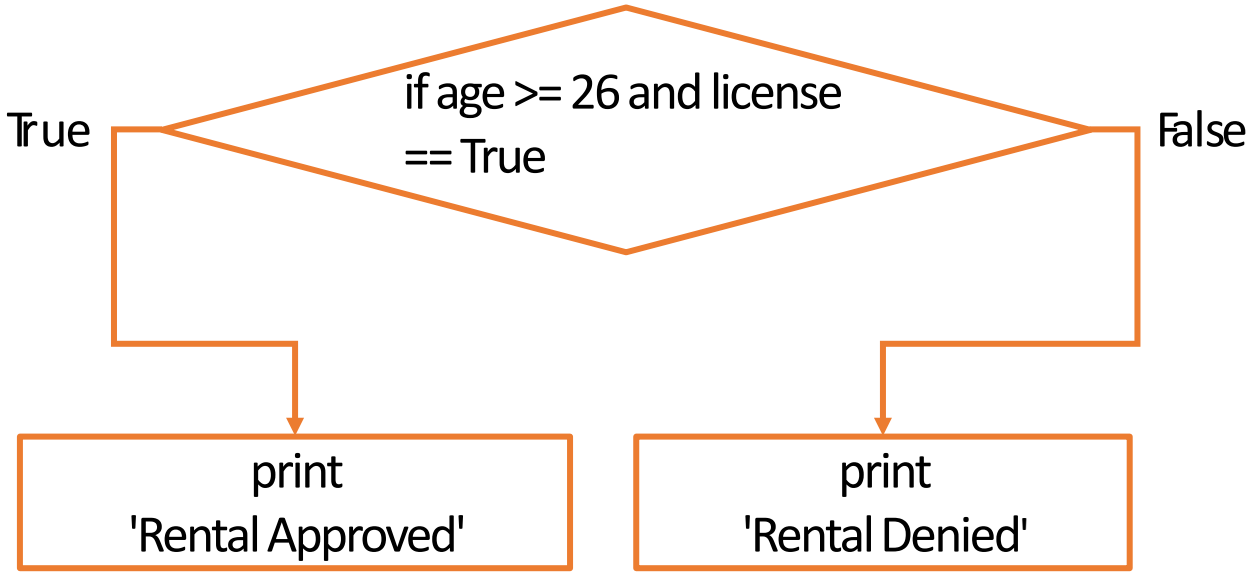


Alternative Car Rental Code

In the code below, we accomplish the same result with the and operation.

This won't always work, though – it depends on how many different results you want.

```
if age >= 26 and license == True:
        print("Rental Approved")
else:
        print("Rental Denied")
```



Nesting and If/Elif/Else Statements

When we have nested conditionals with elif or else clauses, Python pairs them with the if clause at the same indentation level. This is true even if an inner if statement occurs between the outer clauses! However, an outer if/elif/else statement cannot come between parts of an inner conditional.

```
if first == True:
        if second == True:
                print("both true!")
else:
        print("first not true")
```

Question: if we want to add an elsestatement to the inner if, where should it go?

Nesting Conditionals in Functions

When we nest a conditional inside a function definition, we can return values early instead of only returning on the last line. Returning early is fine as long as we ensure every possible path the function can take will eventually return a value.

A function will always end as soon as it reaches a `return` statement, even if more lines of code follow it. For example, the following function will not crash when `n` is zero.

```
def findAverage(total, n):

    if n <= 0:
        return "Cannot compute the average"
    return total / n
```

Exercise: Convert Flow Chart to Code