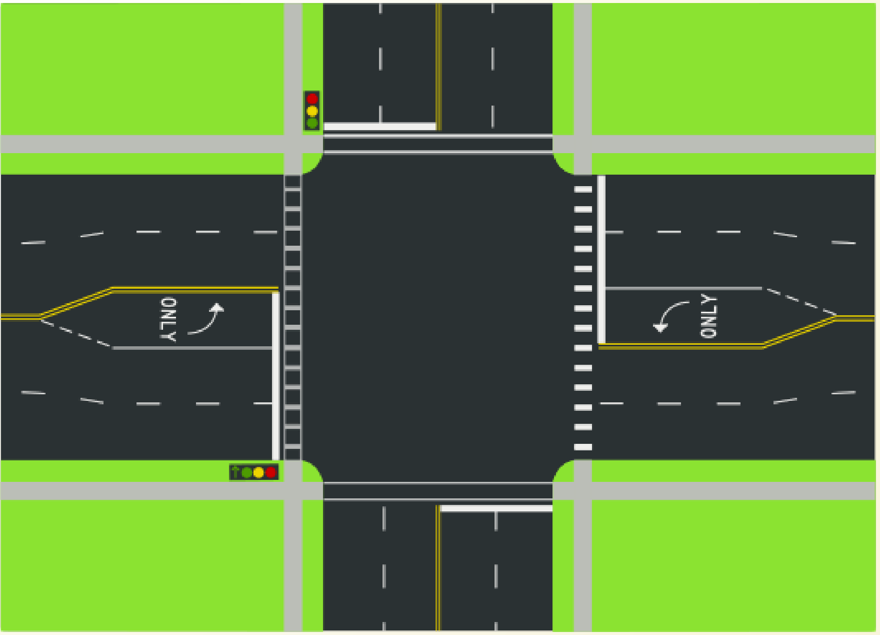
Conditionals and Boolean Logic

adapted from material by Mike Scott and Bill Young at the University of Texas at Austin

So far, we've looked at how to run *straight-line* code

Do A, then do B, then do C

But often, we need to ask a question and do something different based on the answer.



Do I need to stop?

Sneak Peek

```python
if light == "red":
  stop()
elif light == "green":
  go()
elif light == "yellow":
  slow_down()
else:
  # Note: don't ever do this
  crash_car()
```

Booleans

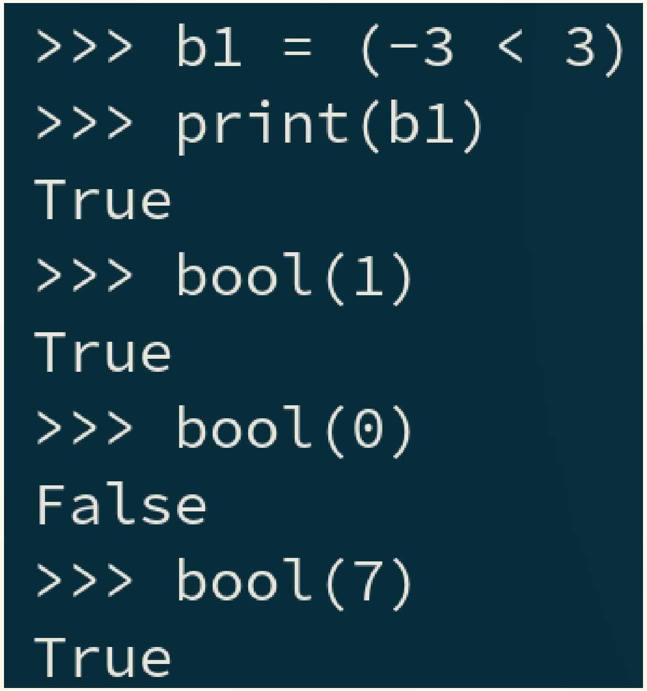
Boolean values are a useful way to refer to the answer to a yes/no question.

The boolean values are True and False.

```
>>> import math
>>> b = ( 30.0 < math.sqrt(1024))
>>> print(b)
True
>>> x = 1
>>> x < 0
False
>>> x >= -2
True
>>> b = (x == 0)
>>> print(b)
False
```

Boolean Representation

Internally, Python represents False as 0 and True as 1. You can convert back and forth using the bool and int functions.

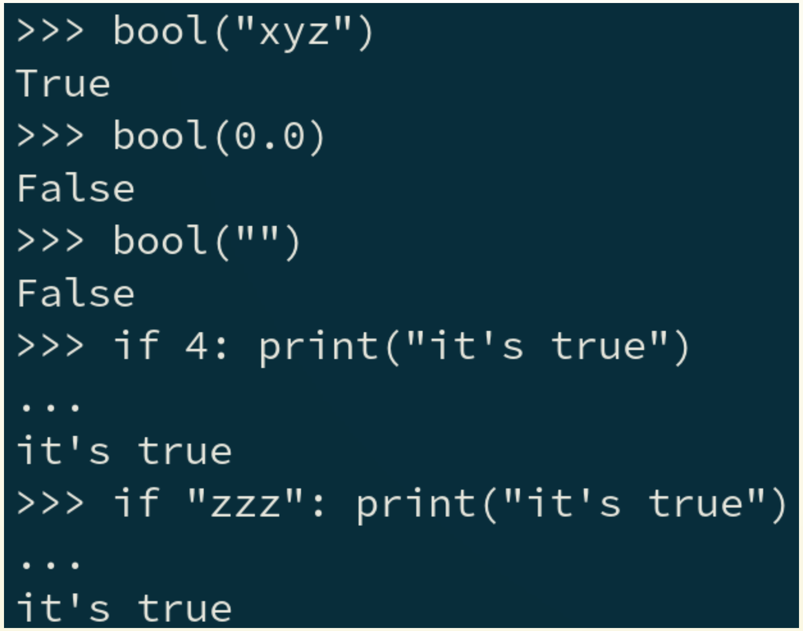


```
>>> b1 = (-3 < 3)
>>> print(b1)
True
>>> bool(1)
True
>>> bool(0)
False
>>> bool(7)
True
```

Boolean Contexts

A **boolean context** is a place where a boolean value is expected.

Within boolean contexts, False, 0, and "" (the empty string), and None are all considered False, and anything else is true. (So-called *truthiness*)



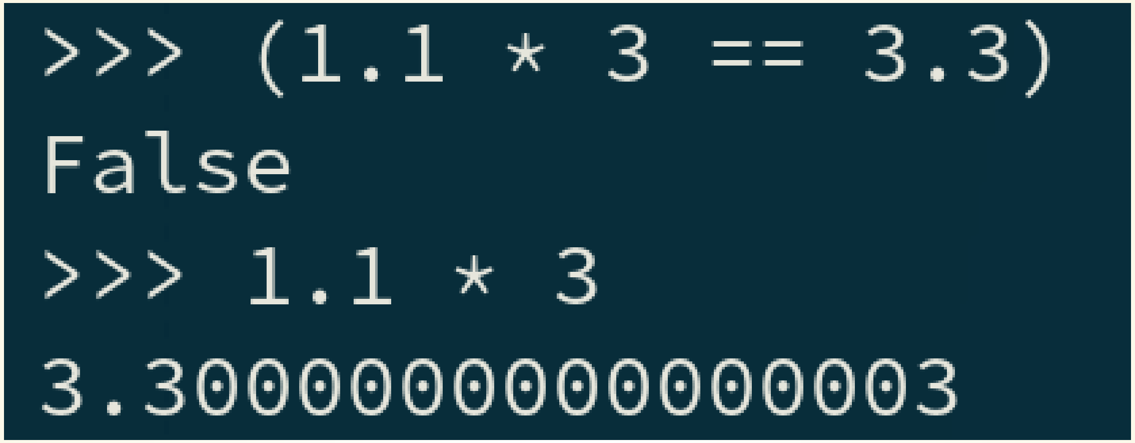
```
>>> bool("xyz")
True
>>> bool(0.0)
False
>>> bool("")
False
>>> if 4: print("it's true")
...
it's true
>>> if "zzz": print("it's true")
...
it's true
```

Comparison Operators

The following comparison operators are useful for comparing numeric values

Operator	Meaning	Example
<	Less than	x < 0
<=	Less than or equal	x <= 0
>	Greater than	x > 0
>=	Greater than or equal	x >= 0
==	Equal to	x == 0
!=	Not equal to	x != 0

Floating Points



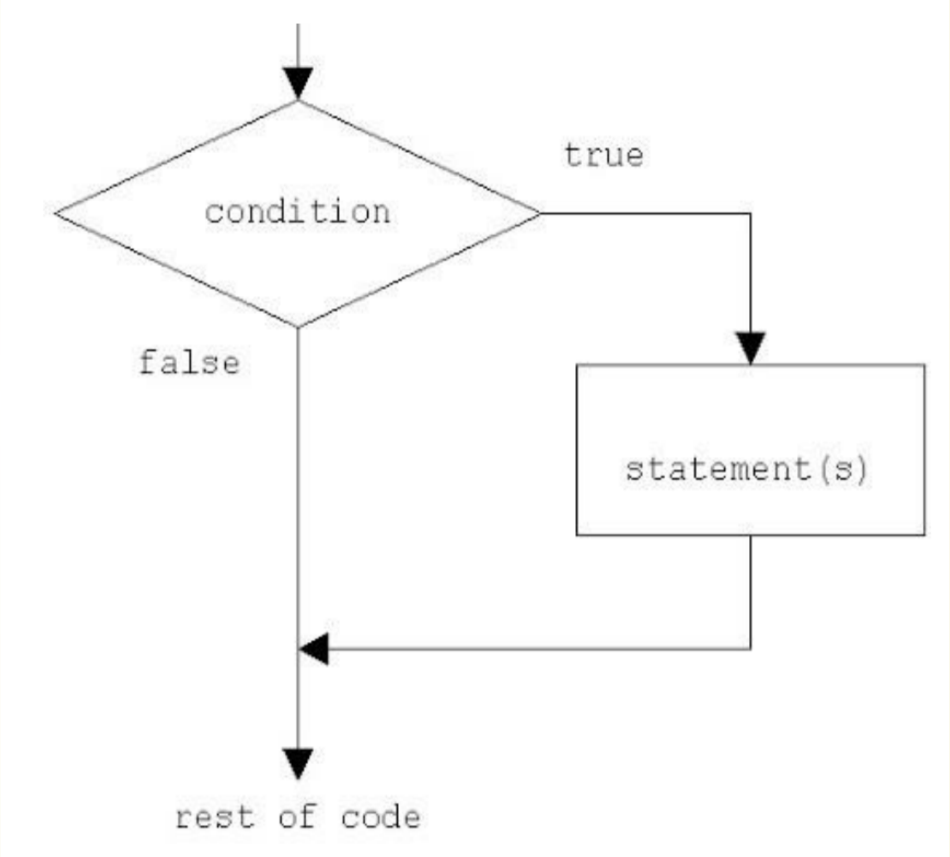
```
>>> (1.1 * 3 == 3.3)
False
>>> 1.1 * 3
3.3000000000000003
```

Remember that floating-point math is *approximate*. This means that some numbers can't be represented perfectly. 3.3 is one of these numbers.

One-Way If Statement

Sometimes we want to perform an action *only if* condition is true.

```
if boolean_expression:
    statement1
    statement2
    # etc
```



Note the colon after the boolean expression.

All of the statements controlled by the if must be indented by the same amount.

Let's Write a Program

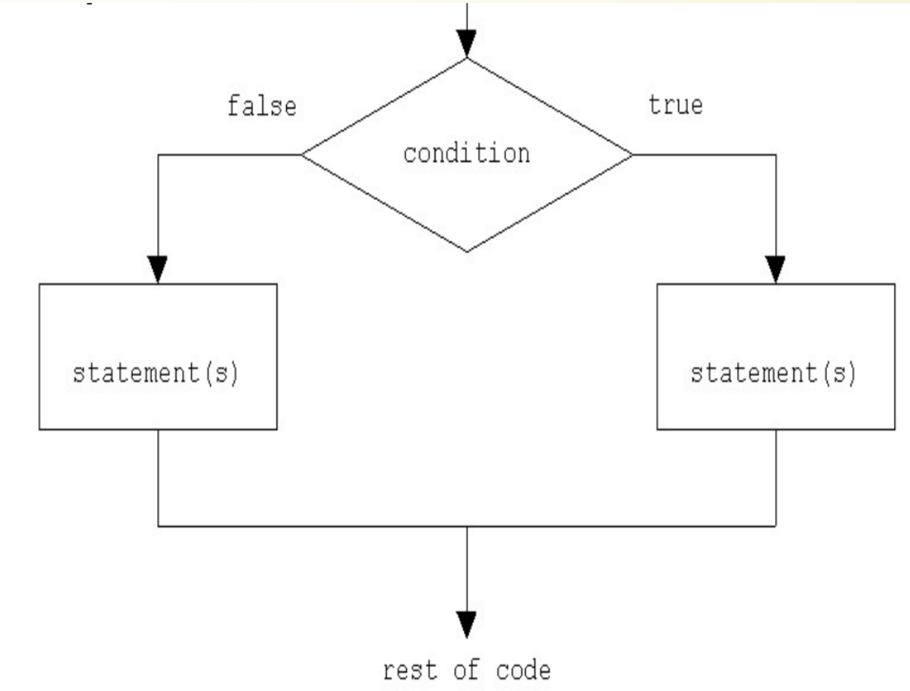
Program will take an input from the user. If the number is zero, do nothing. If the number is nonzero, tell the user what number they entered.

Would if x: work instead of if x != 0?

Two-way If-else

Executes a one of two actions, depending on the value of the boolean expression

```
if boolean_expression:
    true_case_1
    true_case_2
else:
    false_case_1
    false_case_2
```



Notice colons on end of line for both if and else. All the statements in both if and else should be indented the same amount.

Let's Write a Program

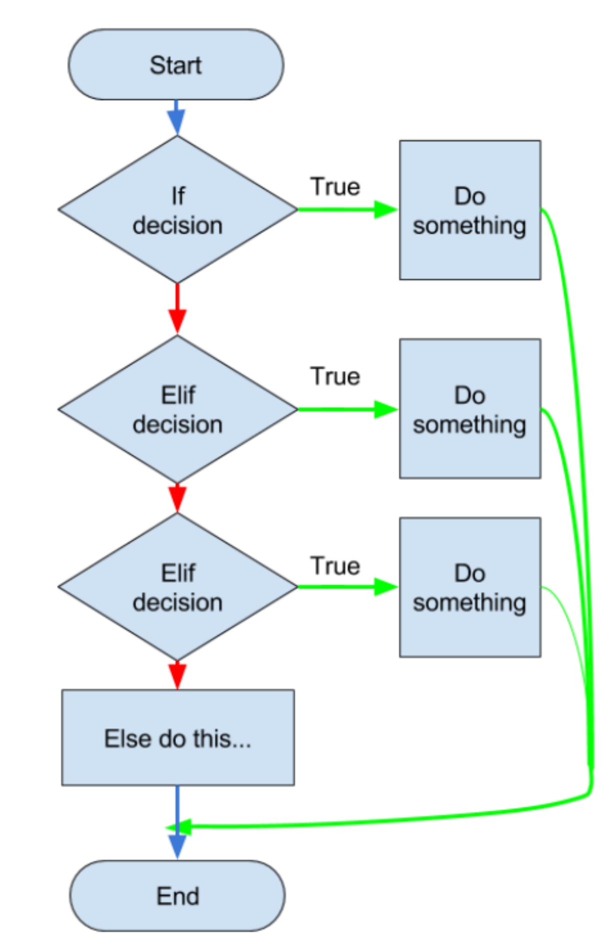
Ask the user for the radius of a circle, then print the area of the circle.

If the radius is negative, tell the user.

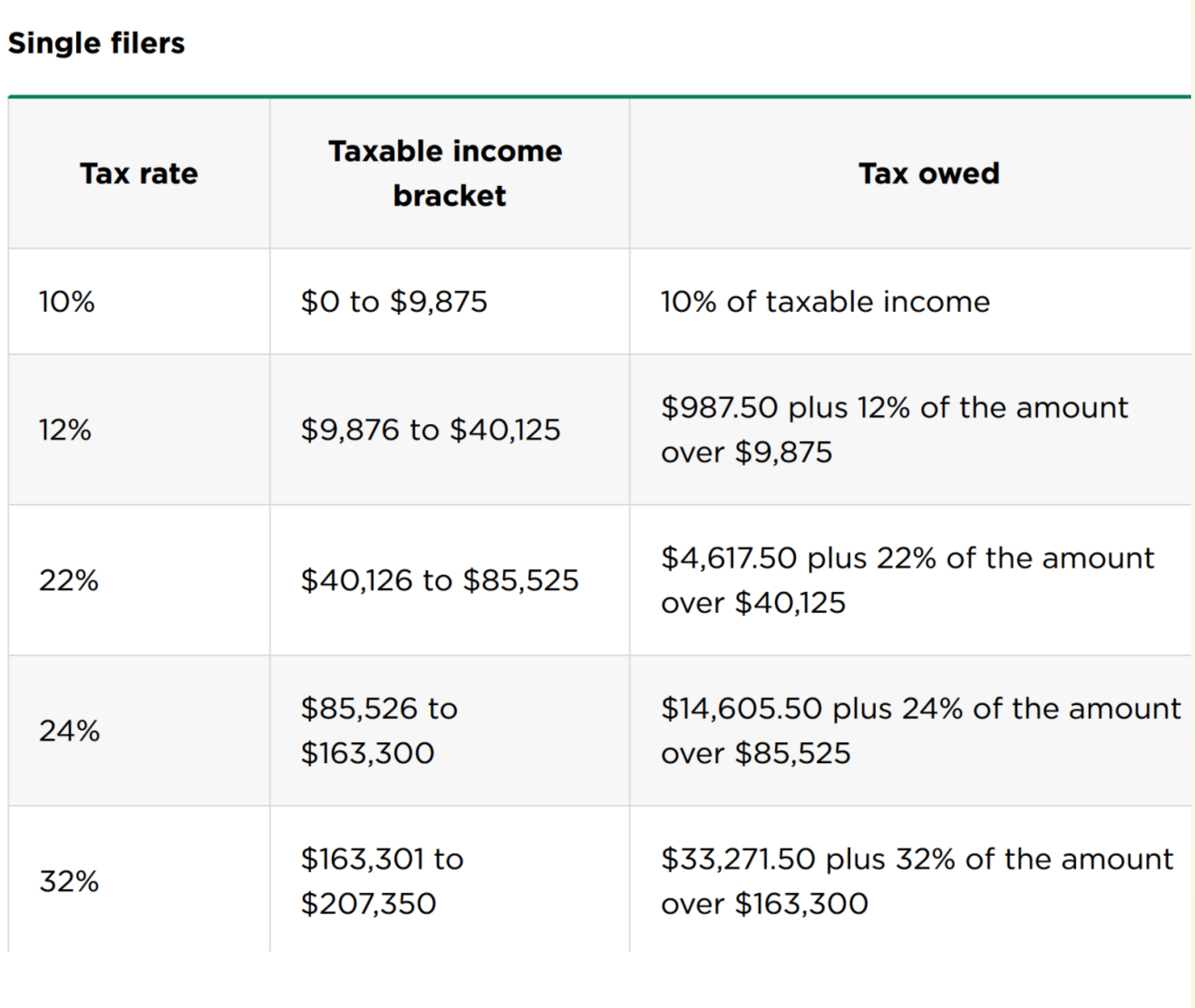
Multi-way Statements

If you have many options, you can use if-elif-else.

```
if boolean_expression_1:
  statement_1
  statement_2
elif boolean_expression_2:
  statement_3
  statement_4
elif boolean_expression_3:
  ...
else:
  statements
```



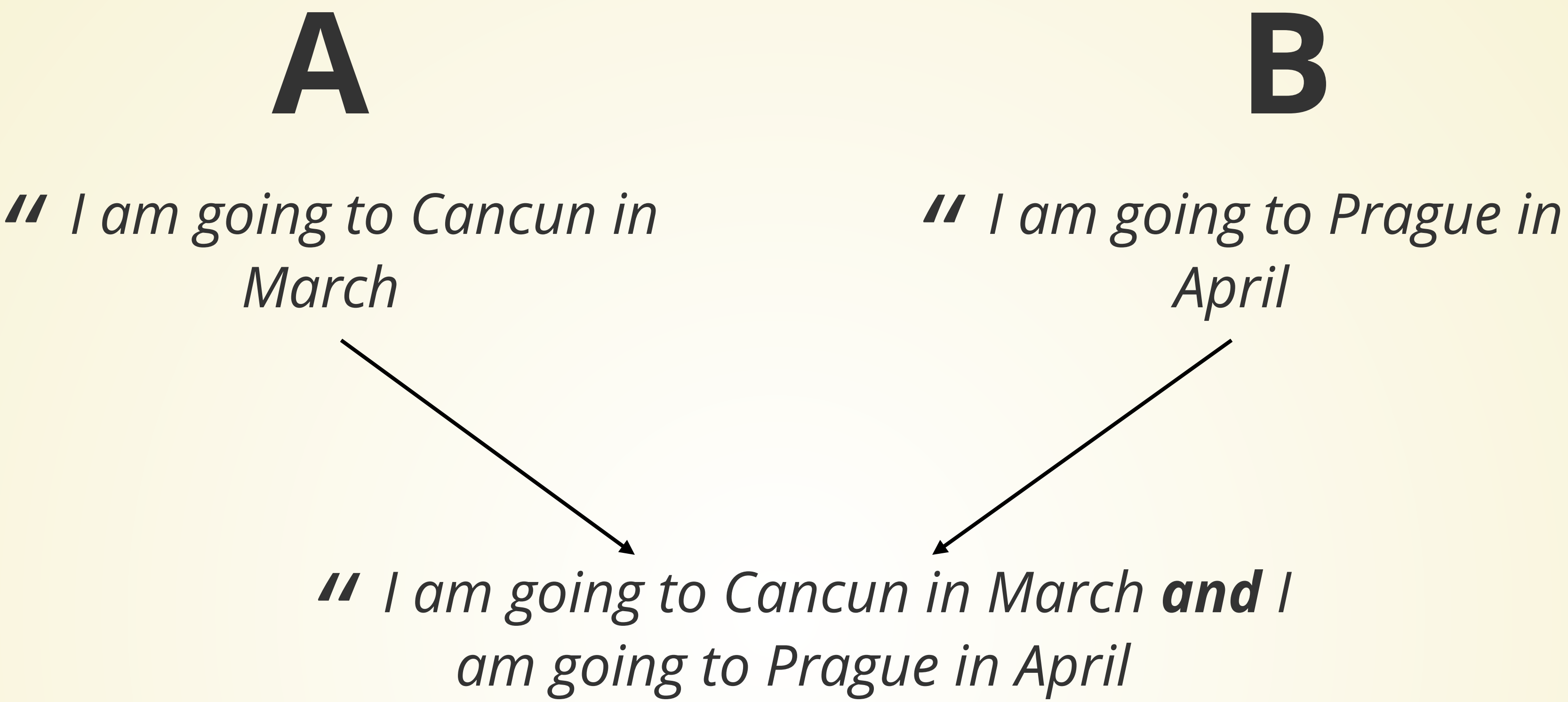
Let's Write a Program



Single filers

Tax rate	Taxable income bracket	Tax owed
10%	$0 to $9,875	10% of taxable income
12%	$9,876 to $40,125	$987.50 plus 12% of the amount over $9,875
22%	$40,126 to $85,525	$4,617.50 plus 22% of the amount over $40,125
24%	$85,526 to $163,300	$14,605.50 plus 24% of the amount over $85,525
32%	$163,301 to $207,350	$33,271.50 plus 32% of the amount over $163,300

Combining Booleans



A

" I am going to Cancun in March

B

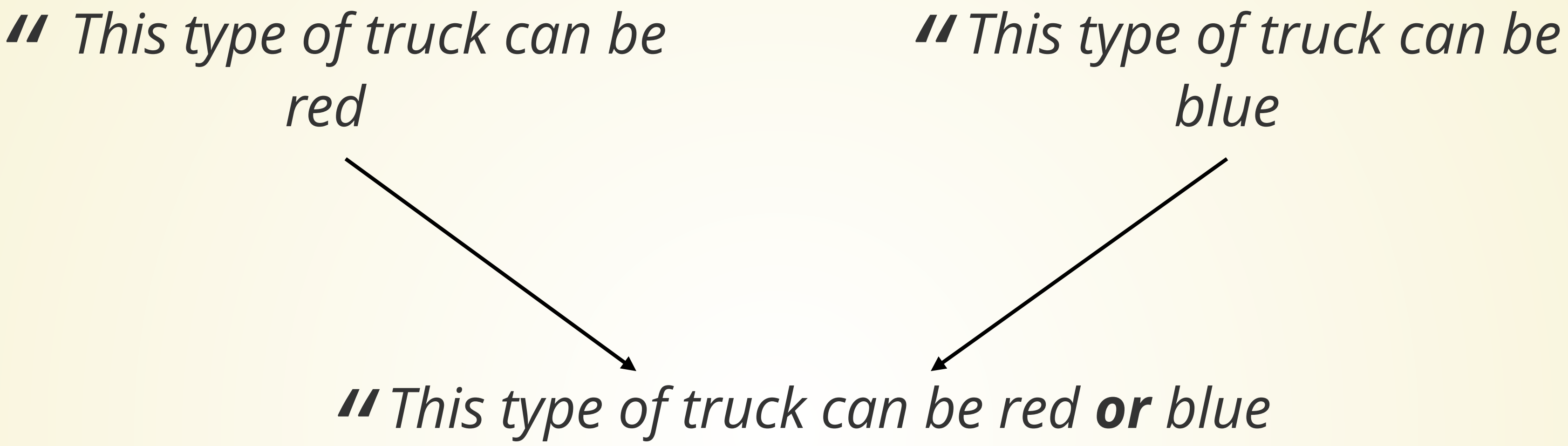
" I am going to Prague in April

" I am going to Cancun in March ***and*** *I am going to Prague in April*

A	**B**	**A and B**
True	True	True
True	False	False
False	True	False
False	False	False

A

B



“ *This type of truck can be red*

“ *This type of truck can be blue*

“ *This type of truck can be red* ***or*** *blue*

A	B	A or B
True	True	True
True	False	True
False	True	True
False	False	False

We can use these logical operators to combine boolean expressions:

AND

A	**B**	**A and B**
True	True	True
True	False	False
False	True	False
False	False	False

NOT

A	**not A**
True	False
False	True

OR

A	**B**	**A or B**
True	True	True
True	False	True
False	True	True
False	False	False

Suppose A is false, and we don't know what B is.

What is A and B?

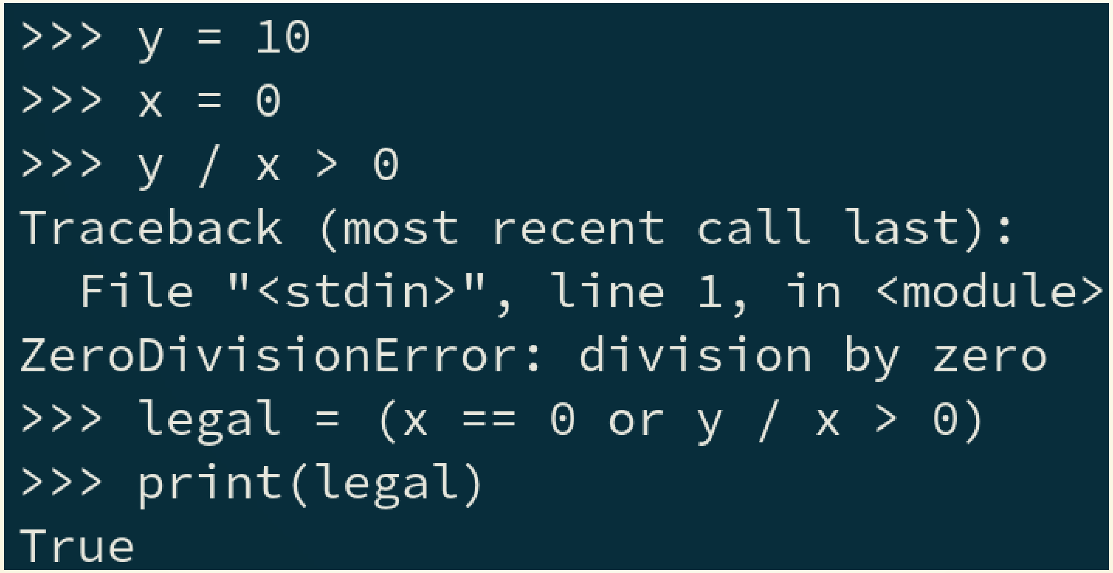
Suppose A is true, and we don't know what B is.

What is A or B?

Short Circuiting

In compound logic expressions, Python will stop as soon as it knows the answer!

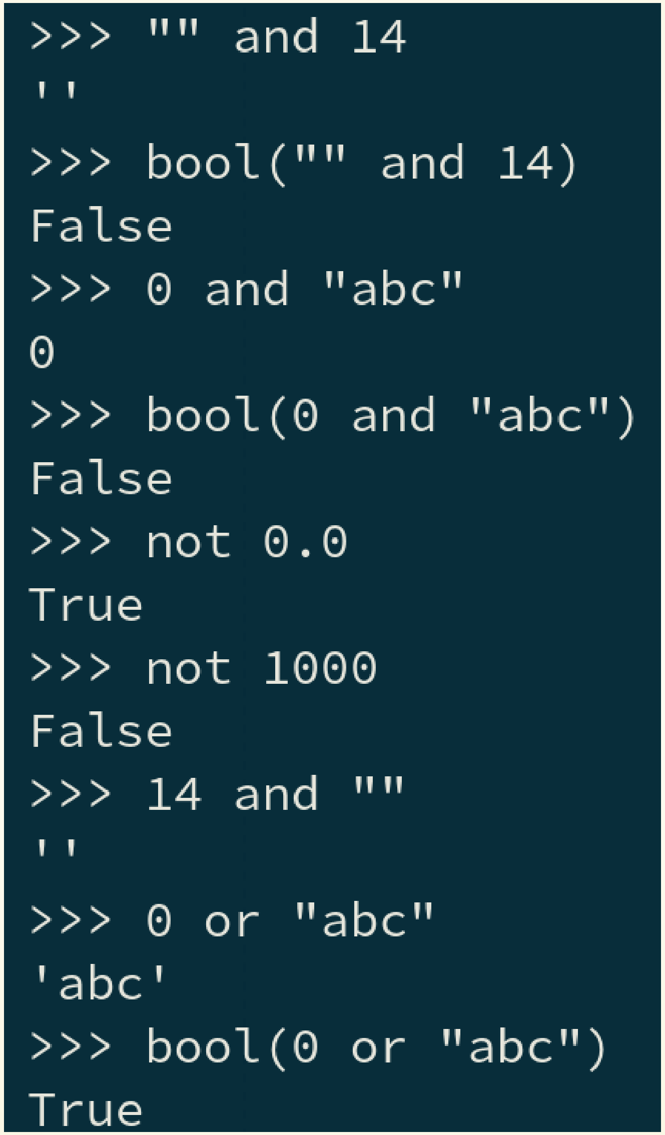
This is known as *short circuiting*, and it sometimes changes how the program runs.



```
>>> y = 10
>>> x = 0
>>> y / x > 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>> legal = (x == 0 or y / x > 0)
>>> print(legal)
True
```

What if we change the first check to y == 0?

Remember: in a Boolean context, Python gives us something boolean-like. **What's going on in each case here?**



```
>>> "" and 14
''
>>> bool("" and 14)
False
>>> 0 and "abc"
0
>>> bool(0 and "abc")
False
>>> not 0.0
True
>>> not 1000
False
>>> 14 and ""
''
>>> 0 or "abc"
'abc'
>>> bool(0 or "abc")
True
```

Leap Years

Julian leap year: every year divisible by 4.

Gregorian Leap Year:

> *Every year that is exactly divisible by four is a leap year, except for years that are exactly divisible by 100, but these centurial years are leap years if they are exactly divisible by 400.*

```python
# Determine if a year entered is a leap year or not
def main():
  year = int(input("Enter a year: "))
  is_leap_year = (year % 4 == 0) and \
                 (not (year % 100 == 0) or (year % 400 == 0))

  if is_leap_year:
    print(year, "is a leap year")
  else:
    print(year, "is not a leap year")

main()
```

Conditional Expressions

A **conditional expression** gives us back one of two values based on a condition.

```
parity = "even" if num % 2 == 0 else "odd"
```

This is equal to:

```
if num % 2 == 0:
    parity = "even"
else:
    parity = "odd"
```

The general form of the expression is:

```
expr_1 if boolean_expr else expr_2
```

which means expr_1 if boolean_expr is True, and expr_2 otherwise.

What does this code do?

```
max_xy = x if x >= y else y
```

Conditional expressions can simplify your code!

```python
# Determine if three numbers are sorted ascending
def main():
  x = float(input("Enter first number: "))
  y = float(input("Enter second number: "))
  z = float(input("Enter third number: "))

  print("Ascending" if x <= y and y <=z else "Not Ascending")
```

Operator Precedence

Sometimes, it can be *ambiguous* as to what an expression means:

3 + 4 * 5

Does this mean

- 7 * 5
- 3 + 20

a and b or c

Does this mean

- (a and b) or c
- a and (b or c)

Precedence rules! For arithmetic, we do multiplication before addition.

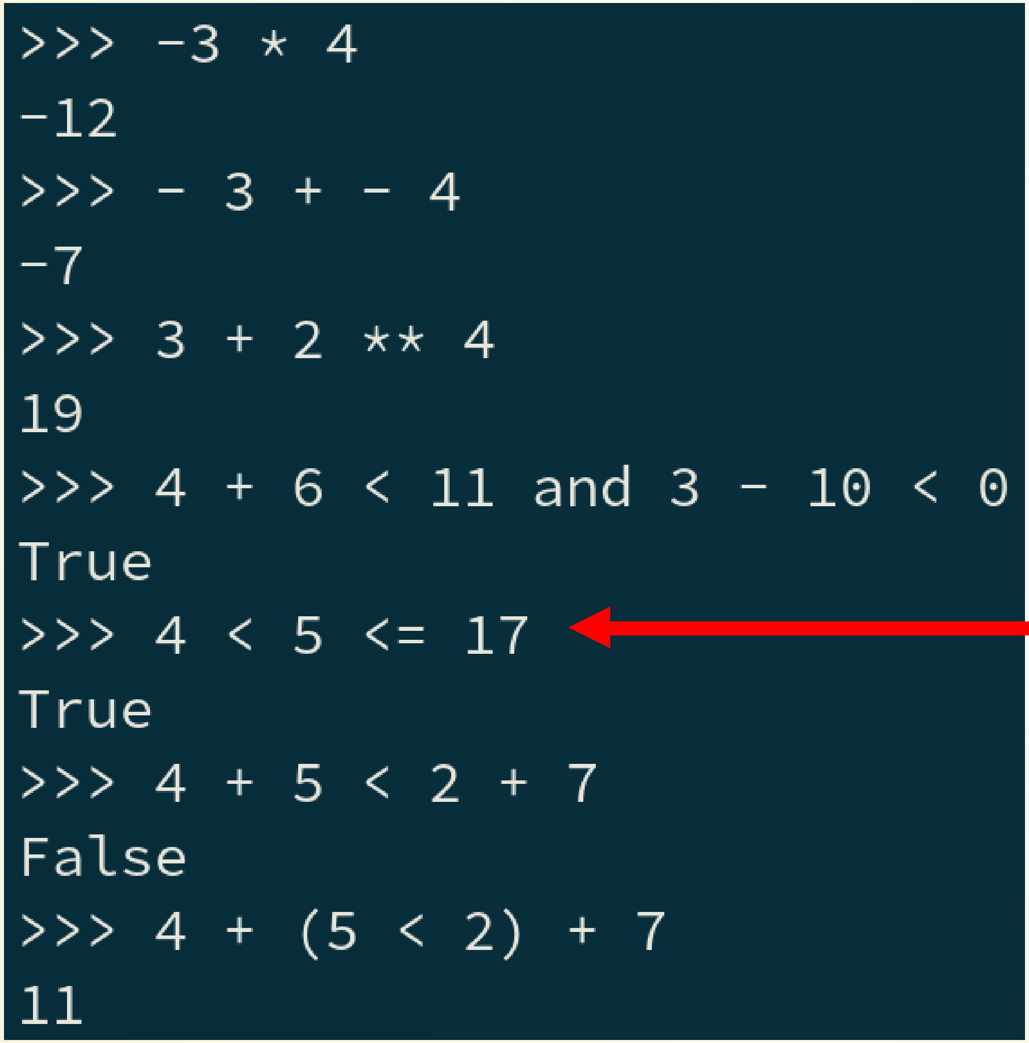
This chart contains the *precedence rules* for Python.

Higher items have higher precedence (are computed 1st).

Operator	**Meaning**
+, -	**Unary** sign, like -3, or +12
**	Exponentiation
not	logical negation
*, /, //, %	Arithmetic multiplication, division and modulus
+, -	Binary (arithmetic) plus, minus
<, <=, >, >=	Comparison
==, !=	Equal and not equal
and	conjunction (logical and)
or	disjunction (logical or)

a and b or c

Precedence Examples



```
>>> -3 * 4
-12
>>> - 3 + - 4
-7
>>> 3 + 2 ** 4
19
>>> 4 + 6 < 11 and 3 - 10 < 0
True
>>> 4 < 5 <= 17
True
>>> 4 + 5 < 2 + 7
False
>>> 4 + (5 < 2) + 7
11
```

Special syntax

Operators on the same line have the same precedence and are evaluated left-to-right.

Example: 2 + 3 - 5 + 8 is

((2 + 3) - 5) + 8

Parentheses

If the default precedence is wrong for what you need, or you want to make things clearer, you can use parentheses.

```
10 - 8 + 5

(10 - 8) + 5           # What python does

10 - (8 + 5)           # Override defaults

5 - 3 * 4 / 2          # What happens?

5 - ((3 * 4) / 2)      # Much better!
```

Always try to make your code as easy to read as possible.