

CSCI 145 · Introduction to Computing · Fall 2026

Conditionals

Lecture 5 — decision-making in code, if/elif/else

Outline

- Review of comparison & logical operators

- The `if`, `elif`, and `else` statements

- Conditionals in functions

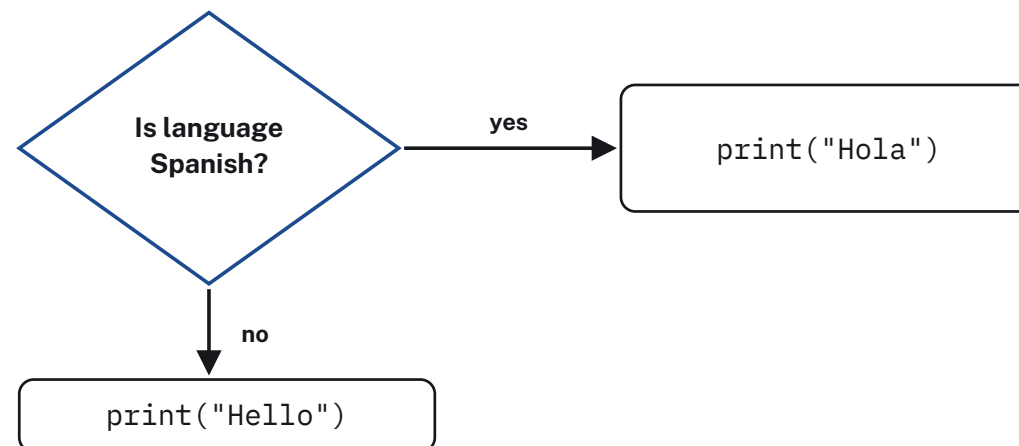
Part 1 of 3

Review of comparison & logical operators

Making decisions in a program

A conditional lets your code do different things depending on whether something is **True** or **False**.

- An alarm clock: **if** the current time equals the wake-up time, ring the alarm
- A greeting program: **if** the user's language is Spanish, print "Hola"; otherwise print "Hello"



Review of comparison operators

These take two values and produce a **boolean**, `True` or `False`.

Operator	Value is <code>True</code> when	Example
<code>a < b</code>	a is less than b	<code>3 < 7</code> is <code>True</code>
<code>a > b</code>	a is greater than b	<code>3 > 7</code> is <code>False</code>
<code>a <= b</code>	a is less than or equal to b	<code>7 <= 7</code> is <code>True</code>
<code>a >= b</code>	a is greater than or equal to b	<code>3 >= 7</code> is <code>False</code>
<code>a == b</code>	a is equal to b	<code>3 == 3</code> is <code>True</code>
<code>a != b</code>	a is not equal to b	<code>3 != 3</code> is <code>False</code>

Review of logical operators

Logical operators combine booleans and produce a new boolean.

Operator	Meaning
a and b	True only if both a and b are True
a or b	True if at least one of a, b is True
not a	True if a is False (flips it)

Evaluating logical expressions

EXERCISE

What does each expression evaluate to?

`9 % 3 != 0 or 8 != 3`

→ False or True → True

`9 % 3 != 0 and 8 != 3`

→ False and True → False

`10 % 2 == 0 and 10 > 0`

→ True and True → True

`not 5 > 8`

→ not False → True

Part 2 of 3

The `if`, `elif`, and `else` statements

The `if` statement

Runs a block of code only when a condition evaluates to `True`.

```
x = 4
if x % 2 == 0:
    print("x is even")
    print(x)
print("The end!")
```

- **Indentation** marks which lines are inside the `if` block
- The first line ends with a `:`

When the condition is True

```
x = 4
if x % 2 == 0:
    print("x is even")
    print(x)
print("The end!")
```

Output:

```
x is even
4
The end!
```

When the condition is **False**

```
x = 3
if x % 2 == 0:
    print("x is even")
    print(x)
print("The end!")
```

Output:

The end!

The `if...else` statement

Choose between **exactly one** of two blocks.

```
x = 4
if x % 2 == 0:
    print("x is even")
else:
    print("x is odd")
print("The end!")
```

If the condition is `False`, the `else` block runs instead of the `if` block — never both, never neither.

Two possible outcomes

```
if x % 2 == 0:  
    print("x is even")  
else:  
    print("x is odd")  
print("The end!")
```

x = 4

`x % 2 == 0` is `True` — the `if` block runs

x is even
The end!

x = 3

`x % 2 == 0` is `False` — the `else` block runs

x is odd
The end!

Nesting conditionals

A conditional's block can contain **another** conditional.

```
if x > 0:
    print("x is positive")
else:
    if x == 0:
        print("x is zero")
    else:
        print("x is negative")
```

This is such a common form that we have a special structure — `elif` — for expressing it so we don't get lost in the increasing indents.

The `if...elif...else` statement

Checks conditions **in order**; runs the block for the first one that's `True`, and skips the rest.

```
x = 104
if x > 0:
    print("x is positive")
elif x == 0:
    print("x is zero")
else:
    print("x is negative")
```

Output:

x is positive

Three possible outcomes

```
if x > 0:  
    print("x is positive")  
elif x == 0:  
    print("x is zero")  
else:  
    print("x is negative")
```

x = 104

x > 0 is True

x is positive

x = 0

x > 0 False, x == 0
True

x is zero

x = -4

both if and elif False

x is negative

Are these equivalent?

Three separate `if` s

```
if x > 0:
    print("x is positive")
if x == 0:
    print("x is zero")
if x < 0:
    print("x is negative")
```

`if` / `elif` / `else`

```
if x > 0:
    print("x is positive")
elif x == 0:
    print("x is zero")
else:
    print("x is negative")
```

ANSWER

For these mutually-exclusive conditions, they print the same thing — but they aren't the same in general. Three separate `if` s always check **every** condition; `elif` stops at the first `True` one. That matters once conditions can **overlap**.

Part 3 of 3

Conditionals in functions

Example: assigning a letter grade

```
def assigned_grade(score):  
    if score >= 90:  
        return 'A'  
    elif score >= 80:  
        return 'B'  
    elif score >= 70:  
        return 'C'  
    elif score >= 60:  
        return 'D'  
    else:  
        return 'F'
```

Takeaway: we can chain as many `elif` s as we need — at most one branch (and one `return`) ever executes.

Ways to compute the absolute value

Compute, then return

```
def absolute1(x):  
    if x < 0:  
        x = x * -1  
    return x
```

Update `x`, then return at the end

If-else, early return

```
def absolute2(x):  
    if x < 0:  
        return x * -1  
    else:  
        return x
```

`return` ends the function immediately — in addition to returning a value to the caller

Implicit else

```
def absolute3(x):  
    if x < 0:  
        return x * -1  
    return x
```

The second `return` is only reached when the condition was `False` — no `else` needed

Activity

1. Download [day05_conditionals.py](#) and save it in your `cs145/class` folder.
2. Launch Thonny. From the File menu, open `cs145/class/day05_conditionals.py`.
3. For part 1, predict what each snippet prints, then uncomment the code one section at a time and check your answers.
4. For part 2, complete `classify_temperature(temp)` using `if / elif / else` to return a description of the weather based on the temperature (in Fahrenheit).
5. If you have time, complete `classify_temperature_and_wind(temp, wind)`, which also takes a wind speed and uses a logical operator to flag "dangerously cold" conditions.

To do

- Review this slide deck and the corresponding notes: [Notes 6](#)
- Exam 1 is this Friday in lab, covering Lectures 1–4
- See the [review materials](#) for a sample exam and reference sheet

Next time

Lecture 6 — iteration