Booleans and Conditionals

Outline

Truth and Falsity

Conditions are an important building block of programs.

We can test a variety of conditions in Python:

```
>>> 1 == 2
False
>>> 1 <= 2
True
>>> "hi" != "hello"
True
```

Operations on Truth Values

- `True` and `False` are special literals in Python.
- They are the only values of type `bool`.
- We have three operations on `bool`:
  - `b1 or b2`: `True` if at least one of `b1` and `b2` is `True`
  - `b1 and b2`: `True` if both `b1` and `b2` are `True`
  - `not b1`: `True` if `b1` is `False`, `False` otherwise

Exercise

Many Boolean expressions are equivalent. Sort these expressions into groups based on whether they mean the same thing.

You may find it useful to consider what each one will produce for the four possible combinations of values for `b1` and `b2`.

1. `b1 and (b2 and b1)`
2. `b1 and b2`
3. `not ((not b1) and (not b2))`
4. `b1 or b2`
5. `(b1 and b2) or (not ((not b1) or (not b2)))`

Branching logic

We can write code that behaves differently depending on Boolean conditions.

```
print("Let's check the number...")
if x > 100:
    print("It's big!")
print("OK, all done")
```

We call `if` a *conditional statement*. Like `def`, it gets a colon and some indented code.

Fancier branching

Any Boolean expression works in an `if`:

```
if (x == "hello") or (x == "hi"):
    print("hello to you too!")
```

More branches

We can also have multiple branches using `elif` ("else if") and a final alternative using `else`:

```
if x < 5:
    print("it's small")
elif x > 100:
    print("really big")
elif x > 10:
    print("kinda big")
else:
    print("pretty mid tbh")
```

What would happen if we swapped the first and second `elif` blocks?

Nested conditionals

The blocks inside of an `if`, `elif`, or `else` can also contain conditional statements:

```
if x == "honest":
    print("Let's see...")
    if y == 10:
        print("Yep, it's 10")
elif x == "liar":
    print("Let's see...")
    if y != 10:
        print("Yep, it's 10")
```

Exercise

This code is out of order and indented wrongly. Please fix it.

```
return "a is bigger"
if a > b:
def compare_sizes(a, b):
return "they're the same"
return "a is smaller"
else:
elif a < b:
```

Exercise

This code lost its conditions somewhere. Please write the correct ones. You can use % to compute a remainder.

```
def is_divisible_by_5_or_10(x):
    if ___:
        return True
    elif ___:
        return True
    else:
        return False
```

Exercise

The function from last time could be simplified into a single Boolean expression (an `if` that does `return True` is often a red flag). We can simplify away the whole if/elif/else in this case. Give it a try!

```
def is_divisible_by_5_or_10(x):
    if ___:
        return True
    elif ___:
        return True
    else:
        return False
```

Quiz

`input`

You'll need to get user input from the terminal for this week's assignment.

- The `input` function asks for input
  - Dual to `print`
- You can give `input` a parameter: `input("> ")`
  - This gives the "prompt" string
- `input` returns the string the user typed before pressing enter

print

`print` outputs values to the console, followed by a newline. We can pass multiple arguments to `print` and they will be joined with spaces:

```
print("hello", 2, "my", "friend")
hello 2 my friend
```

f-strings

Concatenating strings with + can be annoying, especially if you are mixing numbers and strings.
It's often more convenient to use "format strings", also known as f-strings:

```
a = 17
b = 30
c = a + b
print(f"The sum of {a} and {b} is {c}")
```

f-strings start with an `f` before the quotes, and can use curly braces to refer to variables in scope.