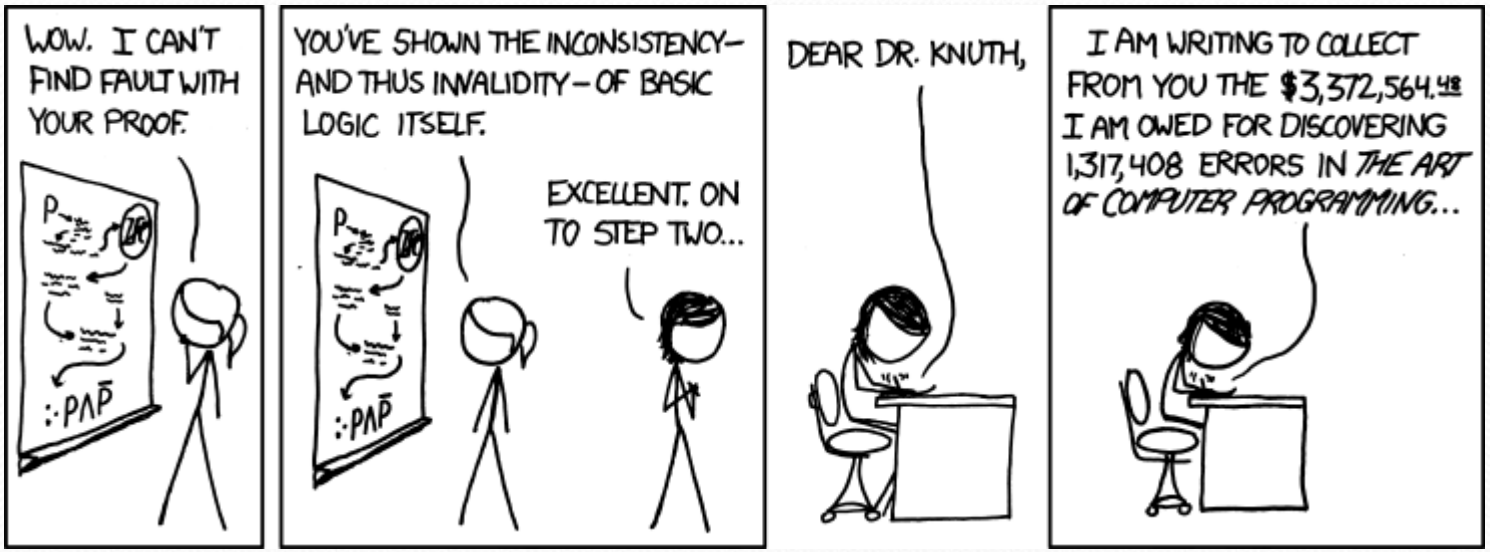
CS 115, Autumn 2021

Lecture 14: `if` / `else` and Boolean logic



Thanks to Marty Stepp and Stuart Reges for parts of these slides

Exercise

- Write a program that prompts the user for the current date and their birthday and prints out which half of the year each is in and if your birthday has occurred yet this year.

Example:

```
What month is today? 5
What day is today? 20
What month is your birthday? 12
What day is your birthday? 10

Today is in the first half of the year.
Your birthday is in the second half of the year.
Your birthday will occur later this year.
```

Nested `if` structures

- exactly 1 path *(mutually exclusive)*

```
if test:
    statement(s)
elif test:
    statement(s)
else:
    statement(s)
```

- 0 or 1 path *(mutually exclusive)*

```
if test:
    statement(s)
elif test:
    statement(s)
elif test:
    statement(s)
```

- 0, 1, or many paths *(independent tests; not exclusive)*

```
if test:
    statement(s)

if test:
    statement(s)

if test:
    statement(s)
```

Which nested `if/else`?

- **(1) if/if/if (2) nested if/else (3) nested if/elif/elif**
  - Whether a user is in the lower, middle, or upper tax bracket on income.
    - **(2)** nested `if / elif / else`
  - Whether you made the dean's list (GPA ≥ 3.8) or honor roll (3.5-3.8).
    - **(3)** nested `if / elif`
  - Whether a number is divisible by 2, 3, and/or 5.
    - **(1)** sequential `if / if / if`
  - Computing a grade of A, B, C, D, or F based on a percentage.
    - **(2)** nested `if / elif / elif / elif / else`

Exercise: bouncing ball

- Draw a ball on a `drawing_panel` and animate its path as it falls.
  - It should bounce when it hits the ground
  - Add a count for how many times the ball bounces
  - Can we make it fall with gravity?
    - displacement = velocity * time + ½ * acceleration * time^2

Relational expressions

- `if` statements use logical tests.

```
if i <= 10: ...
```

  - These are Boolean expressions.
- Tests use *relational operators*:

Operator	Meaning	Example	Value
==	equals	1 + 1 == 2	True
!=	does not equal	3.2 != 2.5	True
<	less than	10 < 5	False
>	greater than	10 > 5	True
<=	less than or equal to	126 <= 100	False
>=	greater than or equal to	5.0 >= 5.0	True

Logical operators

- Tests can be combined using *logical operators*:

Operator	Description	Example	Result
`and`	and	`(2 == 3) and (-1 < 5)`	`False`
`or`	or	`(2 == 3) or (-1 < 5)`	`True`
`not`	not	`not (2 == 3)`	`True`

- "Truth tables" for each, used with logical values *p* and *q*:

p	q	p and q	p or q
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

p	not p
True	False
False	True

Evaluating logical expressions

- Relational operators have lower precedence than math; logical operators have lower precedence than relational operators

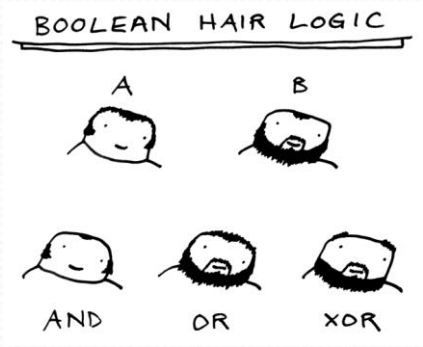
```
5 * 7 >= 3 + 5 * (7 - 1) and 7 <= 11
5 * 7 >= 3 + 5 * 6 and 7 <= 11
35    >= 3 + 30 and 7 <= 11
35    >= 33 and 7 <= 11
True and True
True
```

Logical questions

- What is the result of each of the following expressions?

```
x = 42
y = 17
z = 25
```

  - `y < x and y <= z`
  - `x % 2 == y % 2 or x % 2 == z % 2`
  - `x <= y + z and x >= y + z`
  - `not(x < y and x < z)`
  - `(x + y) % 2 == 0 or not((z - y) % 2 == 0)`

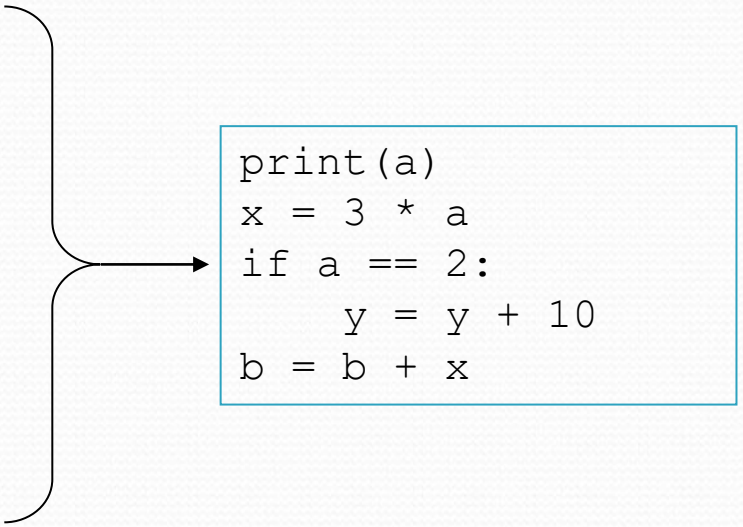


    - Answers: `True`, `False`, `True`, `True`, `False`

Factoring `if/else` code

- **factoring**: Extracting common/redundant code.
  - Can reduce or eliminate redundancy from `if/else` code.
- Example:

```
if a == 1:
    print(a)
    x = 3
    b = b + x
elif a == 2:
    print(a)
    x = 6
    y = y + 10
    b = b + x
else:          # a == 3
    print(a)
    x = 9
    b = b + x
```



```
print(a)
x = 3 * a
if a == 2:
    y = y + 10
b = b + x
```

Type `bool`

- **`bool`**: A logical type whose values are `True` and `False`.
  - A logical ***test*** is actually a Boolean expression.
  - Like other types, it is legal to:
    - create a `bool` variable

```
minor     = age < 21
is_prof   = "Prof" in name
loves_csc = True

# allow only CS-loving students over 21
if minor or is_prof or not loves_csc:
    print("Can't enter the club!")
```

Using `bool`

- Why is type `bool` useful?
  - Can capture a complex logical test result and use it later
  - Can write a function that does a complex test and returns it
  - Makes code more readable

```
good_age    = age >= 27 and age < 39
good_height = height >= 78 and height < 84
rich        = salary >= 100000.0
if (goodAge and goodHeight) or rich:
    print("Okay, let's go out!")
else:
    print("It's not you, it's me...")
```

De Morgan's Law

- **De Morgan's Law**: Rules used to negate boolean tests.
  - Useful when you want the opposite of an existing test.

Original Expression	Negated Expression	Alternative
`a and b`	`not a or not b`	`not(a and b)`
`a or b`	`not a and not b`	`not(a or b)`

  - Example:

Original Code	Negated Code
`if x == 7 and y > 3:` `    ...`	`if x != 7 or y <= 3:` `    ...`