CSCI 141

Lecture 6:
The `bool` data type
Boolean Expressions
Boolean Operators

Happenings

Tuesday, 4/16 – Artificial Intelligence Association Presents: Tableau and Power BI – 6 pm in PH 228

Wednesday, 4/17 – Cybersecurity Lecture Series: Information Security with Austin Tipton – 5 pm in CF 105

Wednesday, 4/17 – Peer Lecture Series: Vim Workshop – 5 pm in CF 162

Thursday, 4/18 – Group Advising to Declare the CS Major – 3 pm in CF 420

Thursday, 4/18 – Western Information Systems Connection – 5 pm in the WWU Library

Announcements

- A2 is out.
  - Due next Monday night
  - Assignments will get more involved as the quarter progresses - start early.

Goals

- Know how to apply operator precedence rules to determine the order in which pieces of an expression are evaluated.

- Understand the use and values of the type `bool` and the meaning of a boolean expression.

- Understand the behavior of the arithmetic comparison operators (`<`, `>`, `<=`, `>=`, `==`, `!=`).

- Understand the behavior of the boolean logical operators `and`, `or`, and `not`

Today’s Quiz

- 2 minutes

Today's Quiz

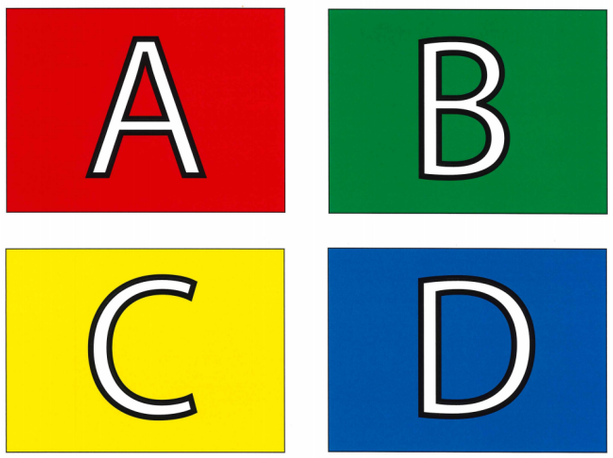
- 2 minutes
- Working with a neighbor: do your answers agree? (2 minutes)

Practice Problem: Operators

Suppose we run the following program, and the user types `6` and presses enter.

What value gets stored in `result`?

```
user_num = input("Enter a number: ")
result = 5 % (3 ** (user_num // 4))
```



A: 1

B: 2

C: 3

D: None of the above

Practice Problem: Operators

Suppose we run the following program, and the user types `6` and presses enter.

What value gets stored in `result`?

```
user_num = input("Enter a number: ")
result = 5 % (3 ** (user_num // 4))
```

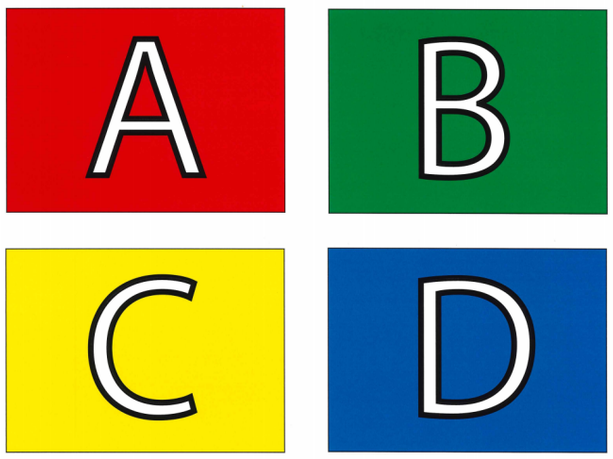
Let's try it out…

Practice Problem: Operators

Suppose we run the following program, and the user types `6` and presses enter.

What value gets stored in `result`?

```
user_num = input("Enter a number: ")
result = 5 % (3 ** (user_num // 4))
```



A: 1

B: 2

C: 3

D: None of the above

Bugs

- We had a bug in our code!
- Why are they called bugs? An anecdote from the history of computing:

September 9th, 1945(!)

Grace Hopper

At 3:45 p.m., Grace Murray Hopper records 'the first computer bug' in the Harvard Mark II computer's log book. The problem was traced to a moth stuck between relay contacts in the computer, which Hopper duly taped into the Mark II's log book with the explanation: “First actual case of bug being found.” The bug was actually found by others but Hopper made the logbook entry.

Source: https://www.computerhistory.org/tdih/september/9/

“First actual case of a bug being found”

9/9

0800 Arctan started
1000 " stopped - arctan ✓ { 1.2700 9.037 847 025
9.037 846 ?95 correct
13"UC (032) MP - MC ~~1.9821449000~~ ~~2.13047641 5 (033)~~ 4.615925059(-2)
(033) PRO 2 2.130476415
correct 2.130676415
Relays 6-2 in 033 failed special speed test
In relay " 11.00 test.
Relays changed

Relay #
2145
Relay 337

1100 Started Cosine Tape (Sine check)
1525 Started Multy Adder Test.

1545 Relay #70 Panel F
(moth) in relay.

First actual case of bug being found.

1700 1630 Arctangent started.
1700 closed down.

Practice Problem: Operators

Suppose we run the following program, and the user types `6` and presses enter.

What value gets stored in `result`?

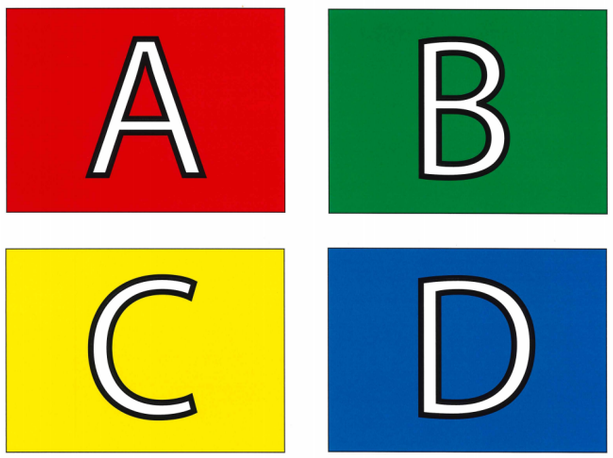
```
user_num = int(input("Enter a number: "))
result = 5 % (3 ** (user_num // 4))
```

Practice Problem: Operators

Suppose we run the following program, and the user types `6` and presses enter.

What value gets stored in `result`?

```
user_num = int(input("Enter a number: "))
result = 5 % (3 ** (user_num // 4))
```



A: 1

B: 2

C: 3

D: None of the above

Practice Problem: Operators

Suppose we run the following program, and the user types `6` and presses enter.

What value gets stored in `result`?

```
user_num = int(input("Enter a number: "))
result = 5 % (3 ** (user_num // 4))
```

Practice Problem: Operators

Suppose we run the following program, and the user types `6` and presses enter.

What value gets stored in `result`?

```
user_num = int(input("Enter a number: "))
result = 5 % (3 ** (user_num // 4))
result = 5 % (3 ** (     6      // 4))
```

Practice Problem: Operators

Suppose we run the following program, and the user types `6` and presses enter.

What value gets stored in `result`?

```
user_num = int(input("Enter a number: "))
result = 5 % (3 ** (user_num // 4))
result = 5 % (3 ** (    6    // 4))
result = 5 % (3 **          1       )
```

Practice Problem: Operators

Suppose we run the following program, and the user types `6` and presses enter.

What value gets stored in `result`?

```
user_num = int(input("Enter a number: "))
result = 5 % (3 ** (user_num // 4))
result = 5 % (3 ** (     6     // 4))
result = 5 % (3 **              1       )
result = 5 % (                  3       )
```

Practice Problem: Operators

Suppose we run the following program, and the user types `6` and presses enter.

What value gets stored in `result`?

```
user_num = int(input("Enter a number: "))
result = 5 % (3 ** (user_num // 4))
result = 5 % (3 ** (     6     // 4))
result = 5 % (3 **           1       )
result = 5 % (               3       )
result =                 2
```

Order of Operations

We know parenthesized expressions get evaluated from inside to out. Are there any other rules?

What if we took the parentheses out:

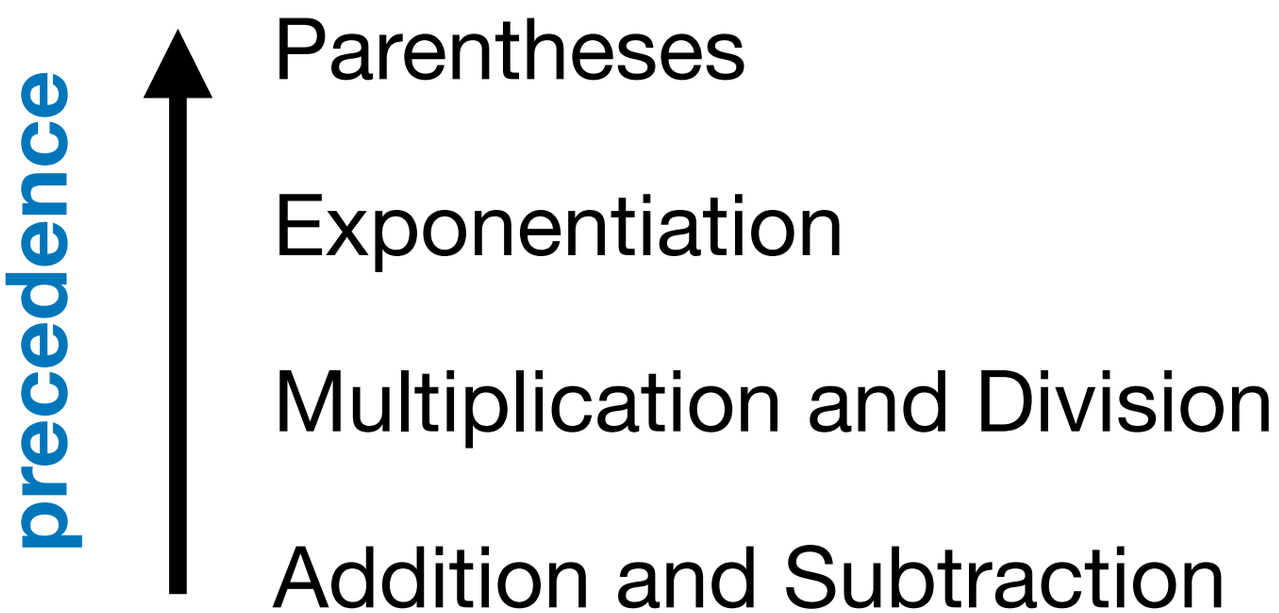
```
result = 5 % (3 ** (6 // 4))
```

```
result = 5 % 3 ** 6 // 4
```

Order of Operations

We know parenthesized expressions get evaluated from inside to out. Are there any other rules? Yes: operator precedence.

Remember PEMDAS? BIDMAS? BODMAS?



Order of Operations

We know parenthesized expressions get evaluated from inside to out. Are there any other rules? Yes: operator precedence.

Remember PEMDAS? BIDMAS? BODMAS?

Parentheses

Exponentiation

Multiplication and Division

Addition and Subtraction

Example:

```
10 * 6 ** 2 / 5 // 11 - 4
```

Order of Operations

We know parenthesized expressions get evaluated from inside to out. Are there any other rules? Yes: operator precedence.

Remember PEMDAS? BIDMAS? BODMAS?

Parentheses

Exponentiation

Multiplication and Division **(left-to-right)**

Addition and Subtraction **(left-to-right)**

Example:

```
10 * 6 ** 2 / 5 // 11 - 4
```

Order of Operations

We know parenthesized expressions get evaluated from inside to out. Are there any other rules? Yes: operator precedence.

Remember PEMDAS? BIDMAS? BODMAS?

Parentheses

Exponentiation

Multiplication and Division (left-to-right)

Addition and Subtraction (left-to-right)

Example:

```
 2 ** 2 ** 4
(2 ** 2) ** 4 = 4⁴ = 16
 2 ** (2 ** 4) = 2¹⁶ = 65536
```

Order of Operations

We know parenthesized expressions get evaluated from inside to out. Are there any other rules? Yes: operator precedence.

Remember PEMDAS? BIDMAS? BODMAS?

Parentheses

Exponentiation **(right-to-left)**

Multiplication and Division (left-to-right)

Addition and Subtraction (left-to-right)

Questions?

Some more familiar operators

`<`	Less than
`>`	Greater than
`<=`	Less than or equal to
`>=`	Greater than or equal to
`==`	
`!=`	

These ones do what you think.

```
3 < 4
4 <= 4
6.7 > 6.3
1000 >= 1000
```

What does `3 < 4` evaluate to?
What does `type(3 < 4)` evaluate to?

We need a new data type!

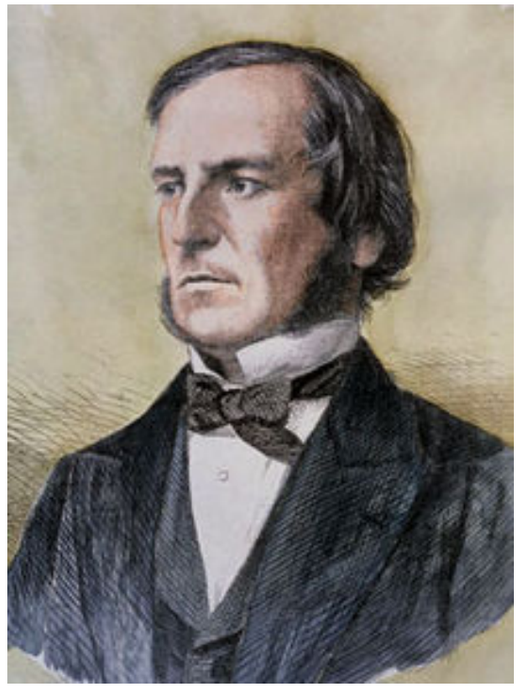
`a < b`

can only be one of two things:
a **true** statement or a **false** statement.

Boolean expressions are expressions that evaluate to one of two possible values: `True` or `False`

What does `3 < 4` evaluate to? `True`
What does `type(3 < 4)` evaluate to? `bool`

The `bool` data type



- Named after 19th century philosopher/ mathematician George Boole, who developed Boolean algebra

- A boolean value (`bool`) represents logical propositions that can be either **true** or **false**.

- In Python, these values are reserved keywords: `True` and `False`. Note capitalization.

- Can be used for things like 3 < 4 or a < b, but also anything else that can be true or false:

```
isRaining = False
```

Some more familiar operators

< Less than

> Greater than

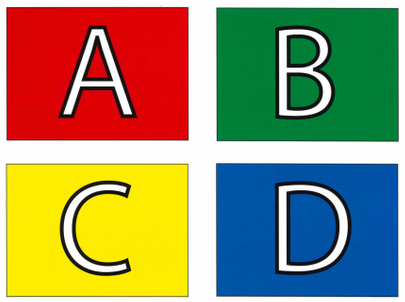
<= Less than or equal to

>= Greater than or equal to

== Equal to

!= Not equal to

What does `3 == 4` evaluate to?



A. False
B. True
C. 7
D. None of the above

Some more familiar operators

`<` Less than

`>` Greater than

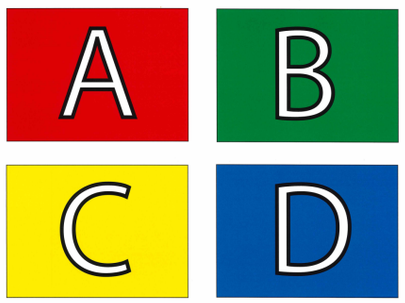
`<=` Less than or equal to

`>=` Greater than or equal to

`==` Equal to

`!=` Not equal to

What does `5 != 4` evaluate to?



A. False
B. True
C. 7
D. None of the above

Some more familiar operators

`<` Less than

`>` Greater than

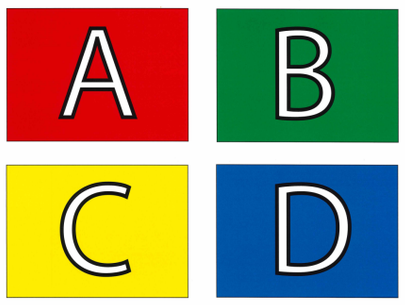
`<=` Less than or equal to

`>=` Greater than or equal to

`==` Equal to

`!=` Not equal to

What does `16 = 4*4` evaluate to?



A. False
B. True
C. 7
D. None of the above

A **classic** mistake:
mixing up = and ==

Some more familiar operators

`<` Less than

`>` Greater than

`<=` Less than or equal to

`>=` Greater than or equal to

`==` Equal to

`!=` Not equal to

`and` logical conjunction, logical and

`or` logical disjunction, logical or

`not` logical negation, logical not

`a and b` is true only when **both** `a` and `b` evaluate to `True`

`a or b` is true when **at least one** of `a` and `b` evaluates to `True`

`not` switches the value:

```
not True => False
not False => True
```

Binary vs Unary Operators

- We have already seen some binary operators and one unary operator.

- Binary operators take two operands:

```
a + b
c // d          etc.
12 != 4
```

- Unary operators take one operand:

```
-b
not False
```

Notice: minus (—) can behave as a unary **or** binary operator!

Truth Tables for `and`, `or`

`x and y`

x \ y	T	F
T	**T**	**F**
F	**F**	**F**

If x is true and y is false, x and y is false.

If `x` is true and `y` is true, `x and y` is true.

Truth Tables for `and`, `or`

`x` **`and`** `y`

x \ y	T	F
T	**T**	**F**
F	**F**	**F**

`x` **`or`** `y`

x \ y	T	F
T	**T**	**T**
F	**T**	**F**

Operator Precedence, Updated

order of precedence

Parentheses

Exponentiation (right-to-left)

Multiplication and Division

Addition and Subtraction

Numerical comparisons `<, >, <=, >=, ==, !=`

`not`

`and`

`or`

order of evaluation

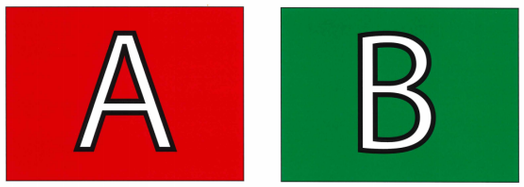
All are evaluated left to right except for exponentiation.

Examples

```
print(3 != 5 and 4 < 7)
                => True and True => True

print(3 == 5 or 4 < 7)
                => False or True => True

print(not False)
                => True

print(3 == 5 or 4 > 7)
                => False or False => False

print(not 6 < 8)
                => not True => False
```

Evaluate This

```
1 == 6 and True or (1.2 < (5 % 3))
```



A. False
B. True
C. 16
D. None of the above

Evaluate This

```
1 == 6 and True or (1.2 < (5 % 3))

1 == 6 and True or (1.2 <     2   )

1 == 6 and True or          True

False  and True or          True

       False      or        True

                  True
```

Next Time: `if` statements

Conditionals: making decisions about what code to execute based on the value of a boolean expression