CSCI 141

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

Announcements

Announcements

- A1 is due tonight!

Announcements

- A1 is due tonight!
    - Please name your files as described. There are 200 of you.

Announcements

- A1 is due tonight!
  - Please name your files as described. There are 200 of you.
    - arithm**a**tic.py **will** break our scripts and require manual handling

Announcements

- A1 is due tonight!
  - Please name your files as described. There are 200 of you.
    - arithm**a**tic.py **will** break our scripts and require manual handling
    - arithmetic-2.py will **not** break our scripts - you can resubmit and the latest version will be graded.

Announcements

- A1 is due tonight!
  - Please name your files as described. There are 200 of you.
    - arithm**a**tic.py **will** break our scripts and require manual handling
    - arithmetic-2.py will **not** break our scripts - you can resubmit and the latest version will be graded.
- Canvas Submission comments: I don't read them. Your TA may not either.

Announcements

- A1 is due tonight!
  - Please name your files as described. There are 200 of you.
    - arithm**a**tic.py **will** break our scripts and require manual handling
    - arithmetic-2.py will **not** break our scripts - you can resubmit and the latest version will be graded.
- Canvas Submission comments: I don't read them. Your TA may not either.
  - If you need to communicate something to me or your TA, send it by email or Canvas message.

Announcements

- A1 is due tonight!
  - Please name your files as described. There are 200 of you.
    - arithm**a**tic.py **will** break our scripts and require manual handling
    - arithmetic-2.py will **not** break our scripts - you can resubmit and the latest version will be graded.
- Canvas Submission comments: I don't read them. Your TA may not either.
  - If you need to communicate something to me or your TA, send it by email or Canvas message.
- A2 will be out tomorrow. Due next Monday night.

Goals

- 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`

QOTD

What does the following program print? Be sure to write the result exactly as Python would print it out.

```
a = 31
b = a // 4
c = (5 % b) - 1.0
print(a, b, c, sep="", end="!")
```

QOTD

The table lists the first few powers of two, their values, and the binary digits in the representation of the decimal number **19**.

- In the second column, fill in the missing powers of two.
- In the third column, fill in the remaining digits of the binary representation of the decimal number 19.

Power	Value	Binary Digit:
2^5		0
2^4		1
2^3	8	
2^2	4	
2^1	2	
2^0	1	

QOTD

The table lists the first few powers of two, their values, and the binary digits in the representation of the decimal number **19**.

- In the second column, fill in the missing powers of two.
- In the third column, fill in the remaining digits of the binary representation of the decimal number 19.

Power	Value	Binary Digit:
2^5		0
2^4	16	1
2^3	8	
2^2	4	
2^1	2	
2^0	1	

QOTD

The table lists the first few powers of two, their values, and the binary digits in the representation of the decimal number **19**.

- In the second column, fill in the missing powers of two.
- In the third column, fill in the remaining digits of the binary representation of the decimal number 19.

Power	Value	Binary Digit:
2^5	32	0
2^4	16	1
2^3	8	
2^2	4	
2^1	2	
2^0	1	

QOTD

The table lists the first few powers of two, their values, and the binary digits in the representation of the decimal number **19**.

- In the second column, fill in the missing powers of two.
- In the third column, fill in the remaining digits of the binary representation of the decimal number 19.

Power	Value		Binary Digit:
2^5	32	*	0
2^4	16	*	1
2^3	8	*	
2^2	4	*	
2^1	2	*	
2^0	+ 1	*	

=

QOTD

The table lists the first few powers of two, their values, and the binary digits in the representation of the decimal number **19**.

- In the second column, fill in the missing powers of two.
- In the third column, fill in the remaining digits of the binary representation of the decimal number 19.

Power	Value	Binary Digit:
2^5	32 *	0
2^4	16 *	1
2^3	8 *	
2^2	4 *	
2^1	2 *	
2^0	+ 1 *	

= **19**

QOTD

The table lists the first few powers of two, their values, and the binary digits in the representation of the decimal number **19**.

- In the second column, fill in the missing powers of two.
- In the third column, fill in the remaining digits of the binary representation of the decimal number 19.

Power	Value		Binary Digit:
2^5	32	*	0
2^4	16	*	1 *(3 left)*
2^3	8	*	
2^2	4	*	
2^1	2	*	
2^0	+ 1	*	
	=		**19**

QOTD

The table lists the first few powers of two, their values, and the binary digits in the representation of the decimal number **19**.

- In the second column, fill in the missing powers of two.
- In the third column, fill in the remaining digits of the binary representation of the decimal number 19.

Power		Value		Binary Digit:
2^5		32	*	0
2^4		16	*	1 *(3 left)*
2^3		8	*	0
2^2		4	*	
2^1		2	*	
2^0	+	1	*	

= **19**

QOTD

The table lists the first few powers of two, their values, and the binary digits in the representation of the decimal number **19**.

- In the second column, fill in the missing powers of two.
- In the third column, fill in the remaining digits of the binary representation of the decimal number 19.

Power		Value		Binary Digit:
2^5		32	*	0
2^4		16	*	1 *(3 left)*
2^3		8	*	0
2^2		4	*	0
2^1		2	*	
2^0	+	1	*	
	=			**19**

QOTD

The table lists the first few powers of two, their values, and the binary digits in the representation of the decimal number **19**.

- In the second column, fill in the missing powers of two.
- In the third column, fill in the remaining digits of the binary representation of the decimal number 19.

Power	Value		Binary Digit:
2^5	32	*	0
2^4	16	*	1 *(3 left)*
2^3	8	*	0
2^2	4	*	0
2^1	2	*	1
2^0	+ 1	*	

= **19**

QOTD

The table lists the first few powers of two, their values, and the binary digits in the representation of the decimal number **19**.

- In the second column, fill in the missing powers of two.
- In the third column, fill in the remaining digits of the binary representation of the decimal number 19.

Power		Value		Binary Digit:
2^5		32	*	0
2^4		16	*	1 *(3 left)*
2^3		8	*	0
2^2		4	*	0
2^1		2	*	1
2^0	+	1	*	1
	=			**19**

QOTD

The table lists the first few powers of two, their values, and the binary digits in the representation of the decimal number **19**.

- In the second column, fill in the missing powers of two.
- In the third column, fill in the remaining digits of the binary representation of the decimal number 19.

Power	Value		Binary Digit:
2^5	32	*	0
2^4	16	*	1 *(3 left)*
2^3	8	*	0
2^2	4	*	0
2^1	2	*	1 *(1 left)*
2^0	+ 1	*	1
	=		**19**

QOTD

The table lists the first few powers of two, their values, and the binary digits in the representation of the decimal number **19**.

- In the second column, fill in the missing powers of two.
- In the third column, fill in the remaining digits of the binary representation of the decimal number 19.

Power	Value		Binary Digit:	
2^5	32	*	0	
2^4	16	*	1	*(3 left)*
2^3	8	*	0	
2^2	4	*	0	
2^1	2	*	1	*(1 left)*
2^0	+ 1	*	1	*(0 left)*
	=		**19**	

QOTD

The third column of the following table contains the binary representation for what decimal number?

Power	Value	Binary Digit:
2^3	8	1
2^2	4	0
2^1	2	1
2^0	1	0

QOTD

The third column of the following table contains the binary representation for what decimal number?

Power	Value	Binary Digit:
2^3	8	* 1
2^2	4	* 0
2^1	2	* 1
2^0	+ 1	* 0

=

QOTD

The third column of the following table contains the binary representation for what decimal number?

Power	Value	Binary Digit:
2^3	8	* 1
2^2	4	* 0
2^1	2	* 1
2^0	+ 1	* 0
	=	10

QOTD

- Suppose the variable a contains a positive integer. Write a single call to the print function that produces the binary representation of $2^a - 1$. For example, if a is 3, the program should print the binary representation of $2^3 - 1 = 7$. You may print the binary representation without any leading zeros.
- Hint: the binary representation of $2^a - 1$ has a special property - try out a few examples of a to get a feel for it.

Last time...

Last time...

- `str + str` performs string concatenation

Last time...

- `str + str` performs string concatenation

  `"Bat" + "man" => "Batman"`

Last time...

- `str + str` performs string concatenation

  `"Bat" + "man" => "Batman"`

- `str * int` performs string repetition:

Last time...

- `str + str` performs string concatenation

    `"Bat" + "man" => "Batman"`

- `str * int` performs string repetition:

    `" toy boat " * 5`

Last time...

- `str + str` performs string concatenation

```
"Bat" + "man" => "Batman"
```

- `str * int` performs string repetition:

```
    " toy boat " * 5
=> "toy boat toy boat toy boat toy boat toy boat"
```

Last time...

- `str + str` performs string concatenation

```
"Bat" + "man" => "Batman"
```

- `str * int` performs string repetition:

```
" toy boat " * 5
=> "toy boat toy boat toy boat toy boat toy boat"
```

- Operator precedence (PEMDAS)

Last time...

- `str + str` performs string concatenation

  `"Bat" + "man" => "Batman"`

- `str * int` performs string repetition:

```
" toy boat " * 5
=> "toy boat toy boat toy boat toy boat toy boat"
```

- Operator precedence (PEMDAS)
- How integers are represented on a computer: Converting between binary and decimal.

I showed you how an `int` is stored.

- What about `str` and `float`?

How do you store strings?

A `str` is a sequence of letters (or characters).

1. Agree by convention on a number that represents each character.
2. Convert that number to binary.
3. Store a sequence of those numbers to form a string.

How do you store strings?

Various conventions exist:
ASCII, Unicode

A `str` is a sequence of letters (or characters).

1. Agree by convention on a number that represents each character.
2. Convert that number to binary.
3. Store a sequence of those numbers to form a string.

How do you store strings?

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
	32	20	[SPACE]	64	40	@	96	60	`
	33	21	!	65	41	A	97	61	a
	34	22	"	66	42	B	98	62	b
	35	23	#	67	43	C	99	63	c
N]	36	24	$	68	44	D	100	64	d
	37	25	%	69	45	E	101	65	e
	38	26	&	70	46	F	102	66	f
	39	27	'	71	47	G	103	67	g
	40	28	(	72	48	H	104	68	h
	41	29	)	73	49	I	105	69	i
	42	2A	*	74	4A	J	106	6A	j
	43	2B	+	75	4B	K	107	6B	k
	44	2C	,	76	4C	L	108	6C	l
	45	2D	-	77	4D	M	109	6D	m
	46	2E	.	78	4E	N	110	6E	n
	47	2F	/	79	4F	O	111	6F	o
	48	30	0	80	50	P	112	70	p
	49	31	1	81	51	Q	113	71	q
	50	32	2	82	52	R	114	72	r
	51	33	3	83	53	S	115	73	s
	52	34	4	84	54	T	116	74	t
DGE]	53	35	5	85	55	U	117	75	u
	54	36	6	86	56	V	118	76	v
K]	55	37	7	87	57	W	119	77	w
	56	38	8	88	58	X	120	78	x
	57	39	9	89	59	Y	121	79	y
	58	3A	:	90	5A	Z	122	7A	z
	59	3B	;	91	5B	[	123	7B	{
	60	3C	<	92	5C	\	124	7C	|
	61	3D	=	93	5D	]	125	7D	}

Decimal	Hex	Char	Decimal	Hex	Char	Decimal
0	0	[NULL]	32	20	[SPACE]	64
1	1	[START OF HEADING]	33	21	!	65
2	2	[START OF TEXT]	34	22	"	66
3	3	[END OF TEXT]	35	23	#	67
4	4	[END OF TRANSMISSION]	36	24	$	68
5	5	[ENQUIRY]	37	25	%	69
6	6	[ACKNOWLEDGE]	38	26	&	70
7	7	[BELL]	39	27	'	71
8	8	[BACKSPACE]	40	28	(	72
9	9	[HORIZONTAL TAB]				73
10	A	[LINE FEED]				74
11	B	[VERTICAL TAB]				75
12	C	[FORM FEED]	44	2C	,	76
13	D	[CARRIAGE RETURN]	45	2D	-	77
14	E	[SHIFT OUT]	46	2E	.	78
15	F	[SHIFT IN]	47	2F	/	79
16	10	[DATA LINK ESCAPE]	48	30	0	80
17	11	[DEVICE CONTROL 1]	49	31	1	81
18	12	[DEVICE CONTROL 2]	50	32	2	82
19	13	[DEVICE CONTROL 3]	51	33	3	83
20	14	[DEVICE CONTROL 4]	52	34	4	84
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85
22	16	[SYNCHRONOUS IDLE]	54	36	6	86
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87
24	18	[CANCEL]	56	38	8	88
25	19	[END OF MEDIUM]	57	39	9	89

this is '`\n`': it's just another character!

That’s how `str` works.

- What about `float`?
- It’s harder to write 4.3752 as a sum of powers of two.

That's how `str` works.

That’s how `str` works.

- Floating-point numbers are stored similarly to scientific notation:

That's how `str` works.

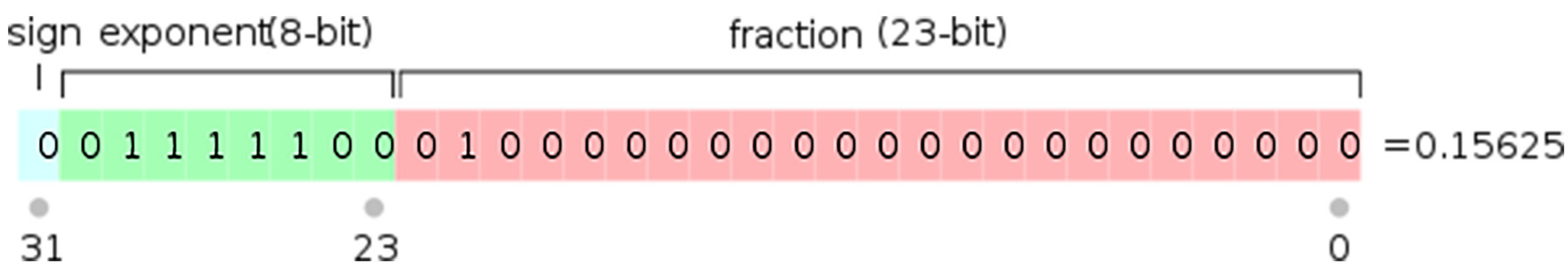
- Floating-point numbers are stored similarly to scientific notation: **$1399.94 = 1.39994 * 10^3$**

That's how `str` works.

- Floating-point numbers are stored similarly to scientific notation: **1399.94 = 1.39994 * 10^3**
- Need to store the base **and** the exponent. In memory, it looks something like this:

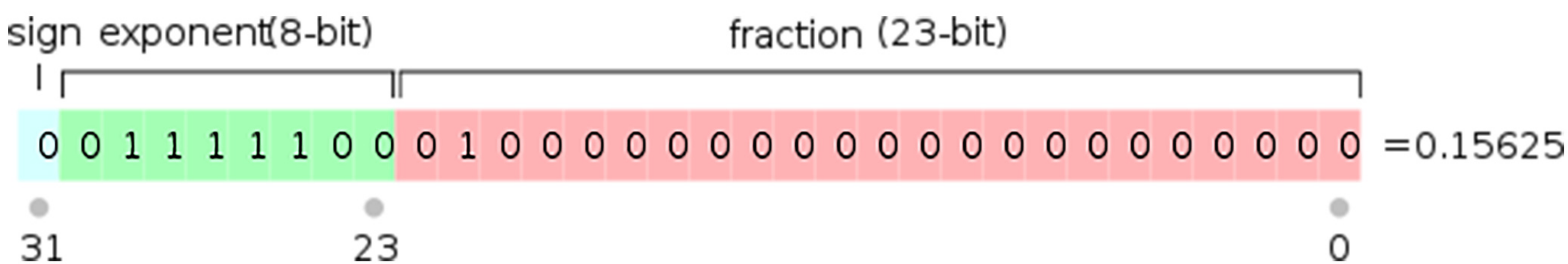
That's how `str` works.

- Floating-point numbers are stored similarly to scientific notation: **1399.94 = 1.39994 * 10^3**
- Need to store the base **and** the exponent. In memory, it looks something like this:



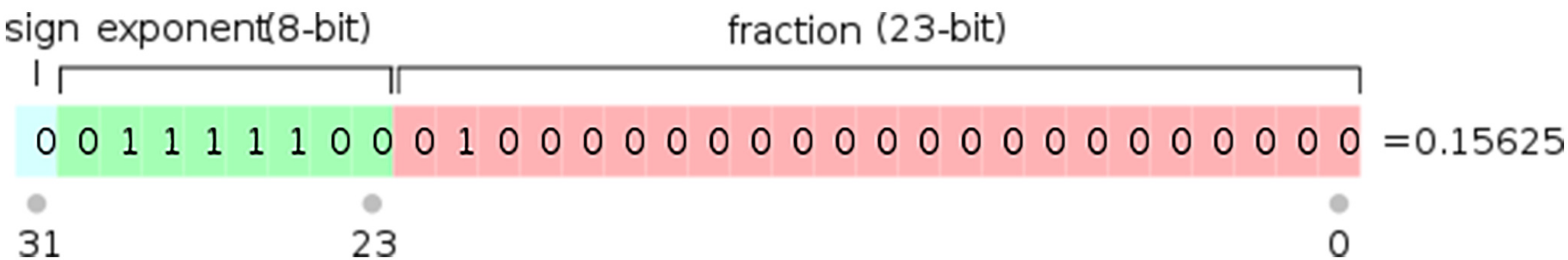
That’s how `str` works.

- Floating-point numbers are stored similarly to scientific notation: **1399.94 = 1.39994 * 10^3**
- Need to store the base **and** the exponent. In memory, it looks something like this:



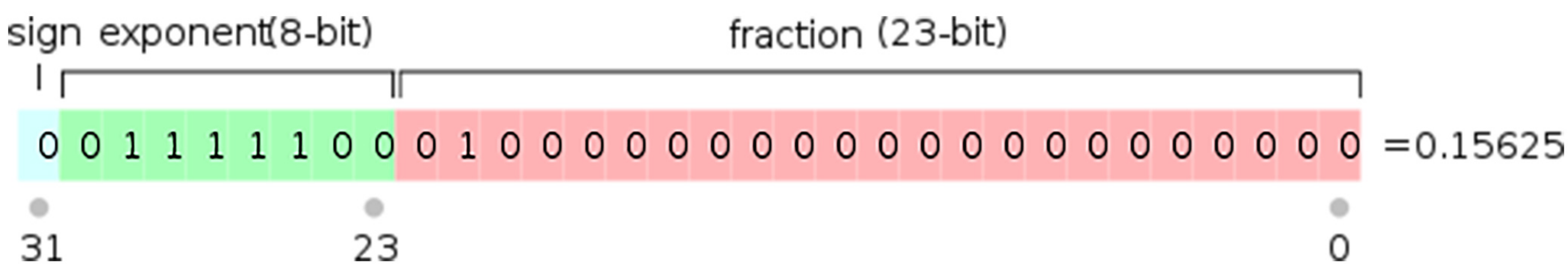
That's how `str` works.

- Floating-point numbers are stored similarly to scientific notation: **1399.94 = 1.39994 * 10^3**
- Need to store the base **and** the exponent. In memory, it looks something like this:



That's how `str` works.

- Floating-point numbers are stored similarly to scientific notation: **$1399.94 = 1.39994 * 10^3$**
- Need to store the base **and** the exponent. In memory, it looks something like this:



- Base and exponent are represented as base-2 integers, so the precision is finite: not all numbers can be represented!

What have we covered so far?

What have we covered so far?

- Data is stored in memory.

What have we covered so far?

- Data is stored in memory.

integers are stored using their *binary* representation

What have we covered so far?

- Data is stored in memory.

  integers are stored using their *binary* representation

- Each piece of data has a type.

What have we covered so far?

- Data is stored in memory.

  integers are stored using their *binary* representation

- Each piece of data has a type.

  so far we've seen: `int, float, str`

What have we covered so far?

- Data is stored in memory.

  integers are stored using their *binary* representation

- Each piece of data has a type.

  so far we've seen: `int, float, str`

- Variables can assign names to pieces of data.

What have we covered so far?

- Data is stored in memory.

  integers are stored using their *binary* representation

- Each piece of data has a type.

  so far we've seen: `int, float, str`

- Variables can assign names to pieces of data. **the assignment operator stores a value in a variable, as in:**

```
my_var = "Hello, world!"
```

What have we covered so far?

- Data is stored in memory.

  integers are stored using their *binary* representation

- Each piece of data has a type.

  so far we've seen: `int, float, str`

- Variables can assign names to pieces of data. **the assignment operator stores a value in a variable, as in:**

  ```
  my_var = "Hello, world!"
  ```

- Operators can do things to the data (these operations are performed by the CPU).

What have we covered so far?

- Data is stored in memory.

  integers are stored using their *binary* representation

- Each piece of data has a type.

  so far we've seen: `int, float, str`

- Variables can assign names to pieces of data. **the assignment operator stores a value in a variable, as in:**

  ```
  my_var = "Hello, world!"
  ```

- Operators can do things to the data (these operations are performed by the CPU).

  so far: assignment operator (=)
  arithmetic operators: (`+,-,*,/,**,//,%`)

What have we covered so far?

What have we covered so far?

- A function can take inputs (arguments) and can produce an output (return value)

What have we covered so far?

- A function can take inputs (arguments) and can produce an output (return value)

- Statements are instructions that are executed

What have we covered so far?

- A function can take inputs (arguments) and can produce an output (return value)

- Statements are instructions that are executed

- Expressions are like phrases that can be evaluated to determine what value they represent.

What have we covered so far?

- A function can take inputs (arguments) and can produce an output (return value)

- Statements are instructions that are executed

  so far: assignment statements, such as `my_var = 64 + 8`

- Expressions are like phrases that can be evaluated to determine what value they represent.

What have we covered so far?

- A function can take inputs (arguments) and can produce an output (return value)

- Statements are instructions that are executed

  so far: assignment statements, such as `my_var = 64 + 8`

- Expressions are like phrases that can be evaluated to determine what value they represent.

  so far:
  - **functions that return values, like `int(42.8)`**
  - **arithmetic expressions, like `(4 + 2) / 2`**
  - **and combinations of other expressions, like `(2**3) // int(user_input)`**

What have we covered so far?

- A function can take inputs (arguments) and can produce an output (return value)

  so far: `input, print, type, int, float, str`

- Statements are instructions that are executed

  so far: assignment statements, such as `my_var = 64 + 8`

- Expressions are like phrases that can be evaluated to determine what value they represent.

  so far:
  - **functions that return values, like `int(42.8)`**
  - **arithmetic expressions, like `(4 + 2) / 2`**
  - **and combinations of other expressions, like `(2**3) // int(user_input)`**

Some more familiar operators

<   Less than

>   Greater than

<= Less than or equal to

>= Greater than or equal to

==

!=

Some more familiar operators

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

==

!=

These ones do what you think.

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
```

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?
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?

We need a new data type!

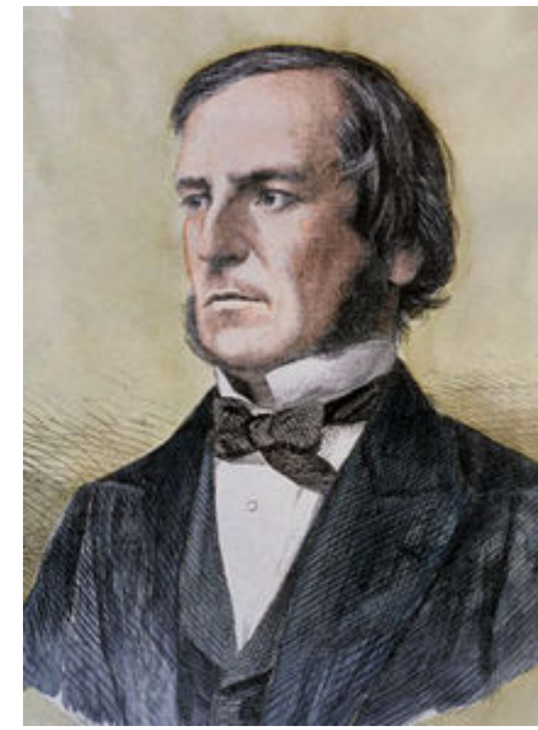
`a < b`

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`

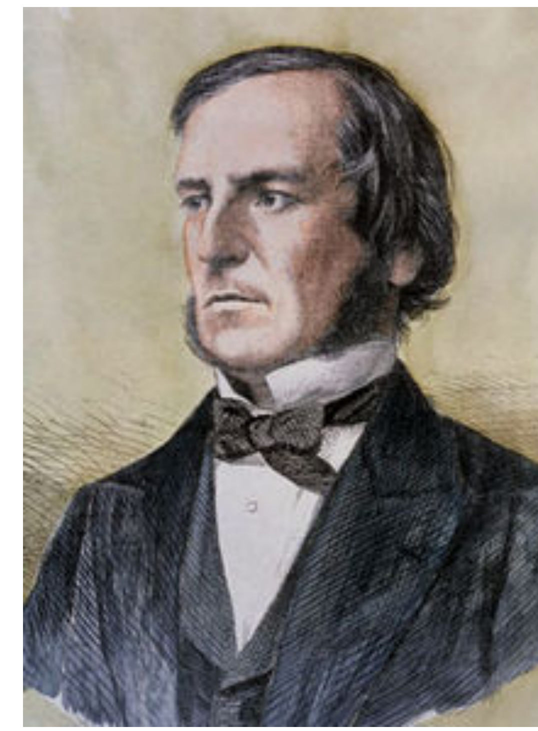
What does `3 < 4` evaluate to? `True`
What does `type(3 < 4)` evaluate to? `bool`

The `bool` data type



The `bool` data type

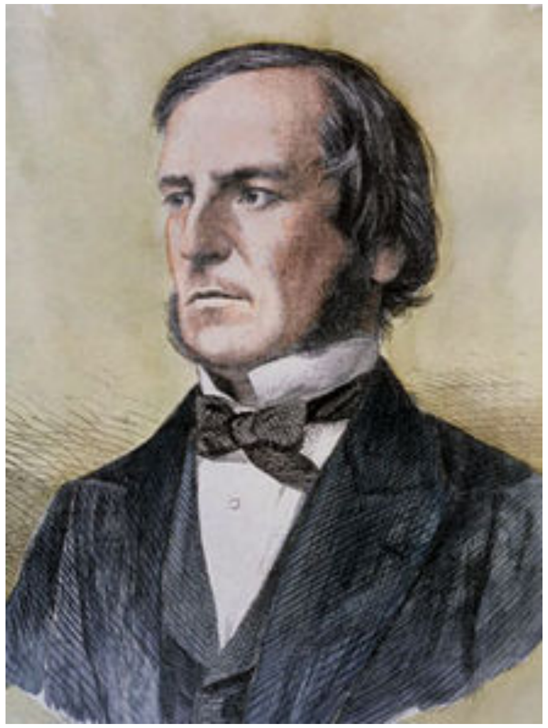
- Named after 19th century philosopher/ mathematician George Boole, who developed Boolean algebra



The `bool` data type

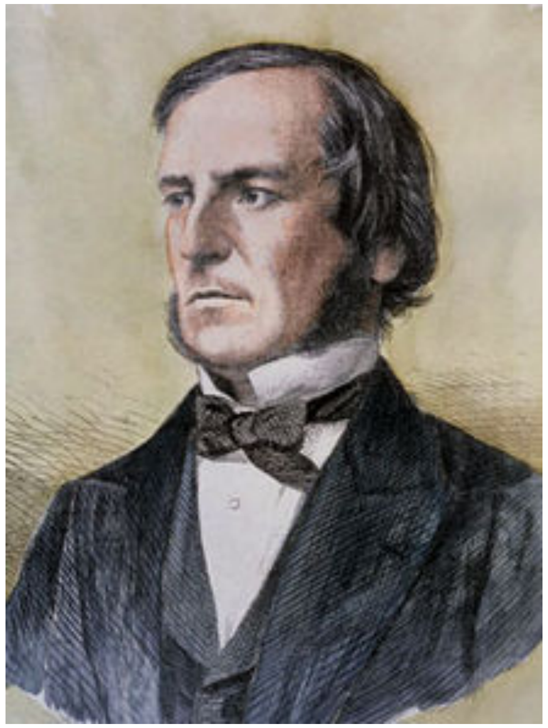
- Named after 19th century philosopher/ mathematician George Boole, who developed Boolean algebra

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



The `bool` data type

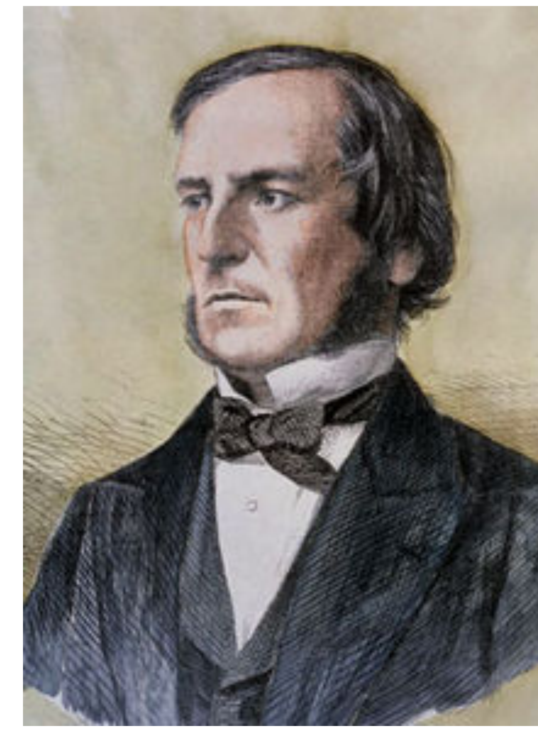
- Named after 19th century philosopher/ mathematician George Boole, who developed Boolean algebra



- 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.

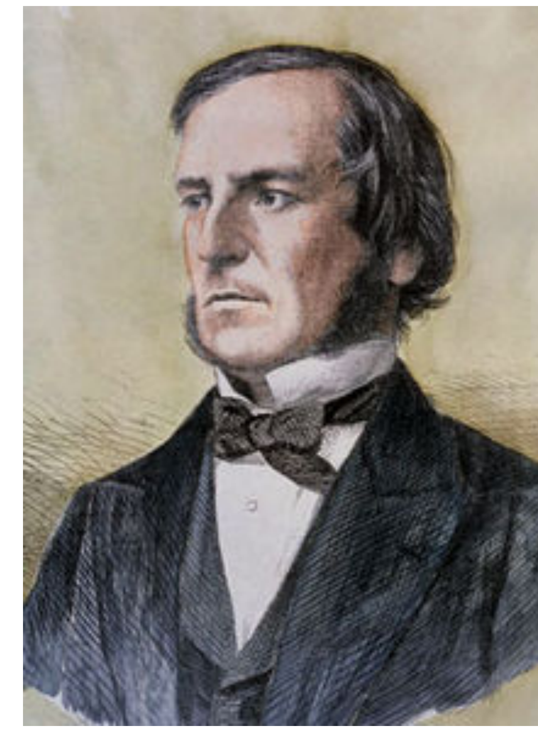
The `bool` data type

- 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:

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:

```
is_raining = False
```

Some more familiar operators

< Less than

> Greater than

<= Less than or equal to

>= Greater than or equal to

== Equal to

!= Not equal to

Some more familiar operators

<   Less than

>   Greater than

<= Less than or equal to

>= Greater than or equal to

== Equal to

!= Not equal to

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

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

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

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`

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`

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
12 != 4
```

etc.

- Unary operators take one operand:

```
-b
not False
```

Binary vs Unary Operators

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

- Binary operators take two operands:

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

etc.

- 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**

Truth Tables for `and`, `or`

`x and y`

		`y`	
		T	F
`x`	T	**T**	**F**
	F	**F**	**F**

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

Truth Tables for `and`, `or`

`x and y`

		y	
		T	F
x	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 (highest at top)

- 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)

print(3 == 5 or 4 < 7)

print(not False)

print(3 == 5 or 4 > 7)

print(not 6 < 8)
```

Examples

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

print(3 == 5 or 4 < 7)


print(not False)


print(3 == 5 or 4 > 7)


print(not 6 < 8)
```

Examples

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

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

print(not False)


print(3 == 5 or 4 > 7)


print(not 6 < 8)
```

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)


print(not 6 < 8)
```

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)
```

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))
```

Evaluate This

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

Evaluate This

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

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

Evaluate This

```
1 == 6 and True or (1.2 < (5 % 3))
1 == 6 and True or (1.2 <    2  )
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

Preview: if statements

Next Time: `if` statements

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