APS106

UNIVERSITY OF TORONTO

Tutorial 3 – Week 4

We’ll be starting at the 10 minute mark

if nothing else, write #cleancode

Agenda

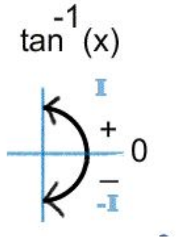
- Lab 2 Review
  - Arctan
- Lecture Review
  - Booleans
  - If-statements
  - String operators
- Practice Questions
- Questions?

Review of Lab 2

arctan

Lab 2 Review- Differences in Arctan Functions

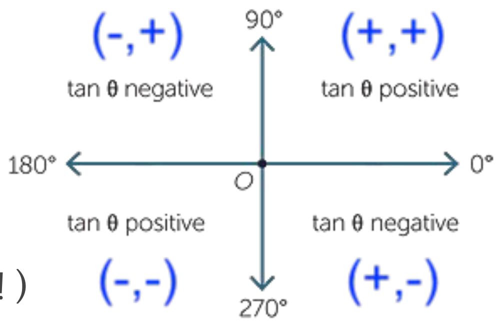
`math.atan(x)`



- Single sign is used in the calculation.

`math.atan2(y, x)`

- Both signs used!!
- Computes correct quadrant for the angle.



Example:

Let p be a point with coordinates (x, y) = (-1, -2)

- `math.atan(-2/-1) = 1.107` (two negative signs cancel out !)
- `math.atan2(-2,-1)= -2.03`

APS106

UNIVERSITY OF TORONTO

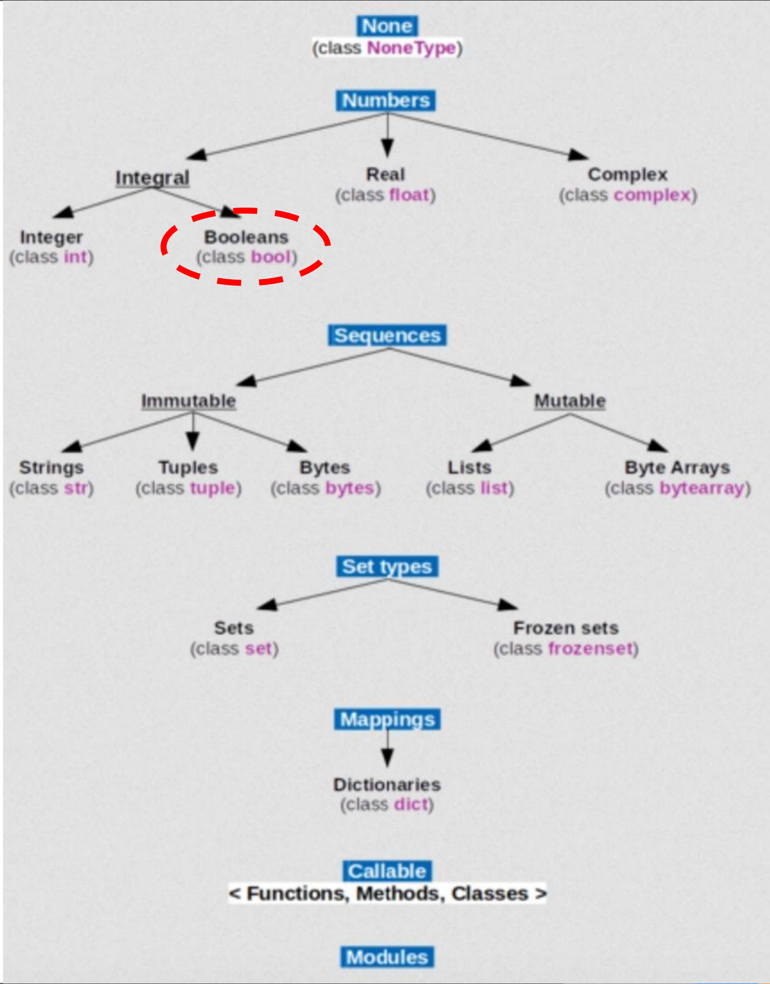
Review of Lecture

Topic 1: Booleans

Lecture Review:
Python Data Types: `bool`

- bool only has two possible values: `True` and `False`

- `bool` is a subtype of integers:
  - `True` is represented internally as 1 and `False` as 0
  - For example, the expression `True * 75` will evaluate to 75, an integer value



Every Python object has a corresponding truth (Boolean) value !

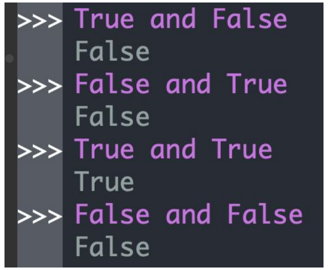
- Use the built-in function **`bool()`** to determine the ***truth value*** of an object.

- `bool()` will return **`False`** when called on the following values:
  - The number zero (`0`)
  - Boolean value `False`
  - `None`
  - An empty string, i.e., `''` or “ ”
  - An empty list , i.e. `[]`
  - An empty tuple, i.e., `()`
  - An empty dictionary, i.e., `{}`

NOTE: the **value** and the **truth value** of an object are not the same thing !!!

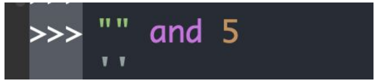
Python Operators: **and**

- When applied to operands of type **bool**, operator **and** functions as the boolean / logic operator **conjunction**. It returns **False** if one of its operands is **False** and returns **True** otherwise.

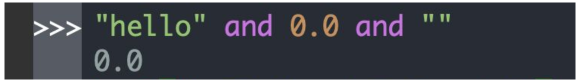


```
>>> True and False
False
>>> False and True
False
>>> True and True
True
>>> False and False
False
```

- When applied to operands of types **other than bool**, operator **and** returns the **value** of the first operand, if the **truth value** of that operand is **False**. Otherwise, it returns the **value** of the second operand.



```
>>> "" and 5
''
```



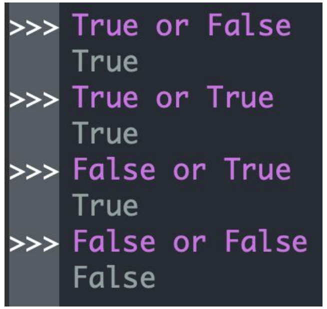
```
>>> "hello" and 0.0 and ""
0.0
```

Explanation:

- **"hello" and 0.0** is evaluated first. Its result is 0.0 (the **value** of the second operand)
- **0.0 and ""** is evaluated next. Its result is 0.0 (the **value** of the first operand).

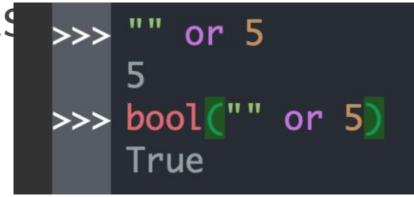
Python Operators: **or**

- When applied to operands of type **bool**, operator **or** functions as the boolean/logic operator **disjunction**. It returns **False** if both of its operands are **False**, and returns **True** otherwise.



```
>>> True or False
True
>>> True or True
True
>>> False or True
True
>>> False or False
False
```

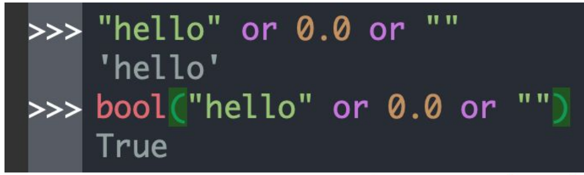
- When applied to operands of types **other than bool**, operator **or** returns the **value** of the first operand, if its **truth value** is **True**. Otherwise, it returns the **value** of the second operand.



```
>>> "" or 5
5
>>> bool("" or 5)
True
```

Explanation:

- **"hello" or 0.0** is evaluated first. Its result is **"hello"** (the **value** of the first operand)
- **"hello" and ""** is evaluated next. Its result is **"hello"** (the **value** of the first operand).

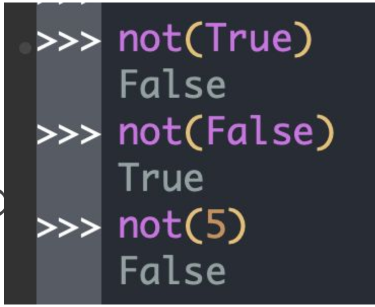


```
>>> "hello" or 0.0 or ""
'hello'
>>> bool("hello" or 0.0 or "")
True
```

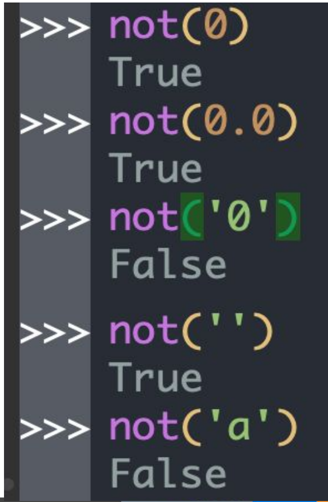
Python Operators: **`not`**

- When applied to operands of type **bool**, operator **`not`** functions as the boolean / logic operator **negation**. It returns **`False`** if its operand is **`True`**, and returns **`True`** if its operand is **`False`**.

- When applied to operands of types **other than `bool`**, operator **`not`** acts as a mapping to Boolean values: it returns **`False`** if the **truth value** of the operand is **`True`**, and returns **`True`** if the **truth value** of the operand is **`False`**.



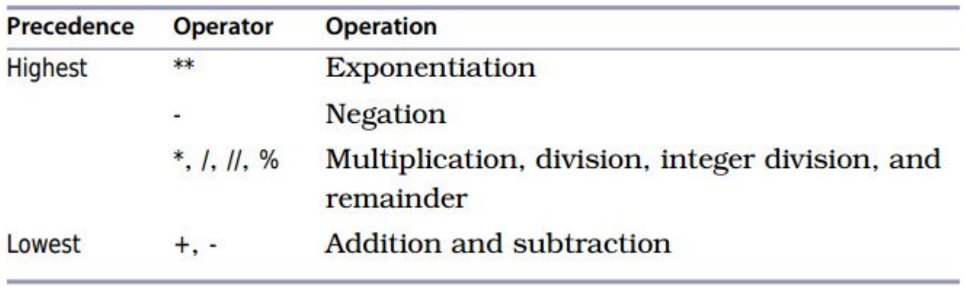
```
>>> not(True)
False
>>> not(False)
True
>>> not(5)
False
```



```
>>> not(0)
True
>>> not(0.0)
True
>>> not('0')
False
>>> not('')
True
>>> not('a')
False
```

Order of Precedence for Python Operators

- Brackets `()` are always first!
- Order of precedence for unparenthesized expressions:
  1. Arithmetic (see table below)
  2. Relational (<=, >, <, ==, etc.) .. More on this soon ☺️
  3. `not`
  4. `and`
  5. `or`



Precedence	Operator	Operation
Highest	**	Exponentiation
	-	Negation
	*, /, //, %	Multiplication, division, integer division, and remainder
Lowest	+, -	Addition and subtraction

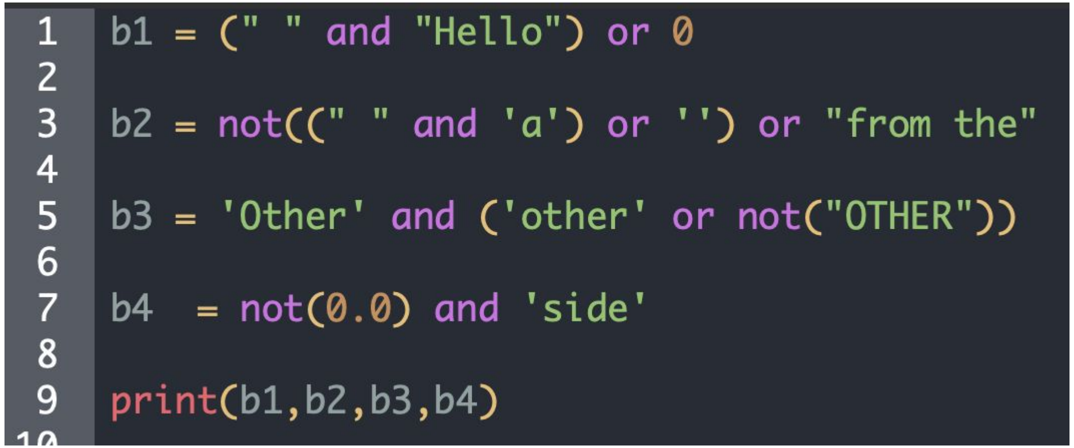
slido

What would this output??

ⓘ Start presenting to display the poll results on this slide.

Practice Problem: Boolean Expressions

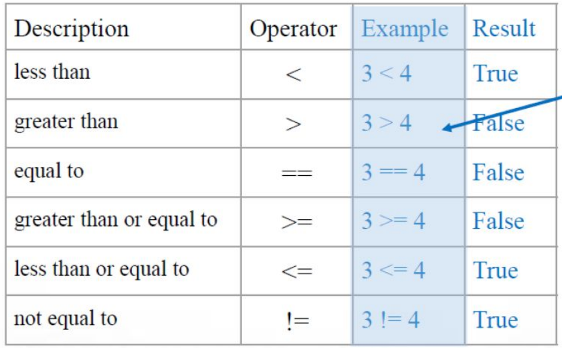
- What would this code output??



```python
b1 = (" " and "Hello") or 0

b2 = not((" " and 'a') or '') or "from the"

b3 = 'Other' and ('other' or not("OTHER"))

b4  = not(0.0) and 'side'

print(b1,b2,b3,b4)
```

Python Comparison Operators

Comparison operators compare two values and produce a `bool` value (either `True` or `False`)



Description	Operator	Example	Result
less than	<	3 < 4	True
greater than	>	3 > 4	False
equal to	==	3 == 4	False
greater than or equal to	>=	3 >= 4	False
less than or equal to	<=	3 <= 4	True
not equal to	!=	3 != 4	True

Boolean Expressions

Warning: There is a **BIG** difference between **=** and **==**

- = is the symbol for the **assignment** operator
- == is the symbol for the "**equal to**" operator

It is a common error to mistakenly use only one = sign when intending to apply the "**equal to**" operator.

Python Comparison Operators Continued

Operator	Description
`is`	Object identity. Returns `True` if both operands are the same object. (we will discuss more about this operator when we discuss Python objects in the future)
`is not`	Negated object identity. Returns `True` if the operands are not the same object. (we will discuss more about this operator when we discuss Python objects in the future)

`in`	Returns `True` if the first operand is contained in the second operand.
	Example:
	`>>> "ti" in "tie"`
	`True`

`not in`	Returns `True` if the first operand is not contained in the second operand. (examples will follow in the next few slides when we talk about strings)
	Example:
	`>>> "te" not in "tie"`
	`True`

Review of Lecture

Topic 2: If-statements

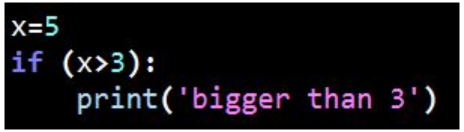
Choice / Conditional Statements: `if`-statements[1]

```
if condition:
    body
```

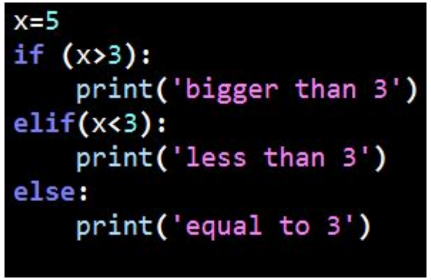
```
if condition1:
    body1
elif condition2:
    body2
elif condition3:
    body3
```

```
if condition1:
    body1
elif condition2:
    body2
...
else: #everything
else
    body3
```

Examples:



```
x=5
if (x>3):
    print('bigger than 3')
```



```
x=5
if (x>3):
    print('bigger than 3')
elif(x<3):
    print('less than 3')
else:
    print('equal to 3')
```

Choice / Conditional Statements: `if`-statements^2

Multiple if-statements

`if`-statements can appear one after another in a program. They are **independent** of each other.

```
if condition1:   # First independent if statement
    body1
if condition2:   # Second independent if statement
    body2
```

Nested if-statements

It is possible to place an IF-statement within the body of another if statement.

```
if condition1:
    if condition2:
        body1
    else:
        body2
```

Equivalent of

Nesting if-statements is not necessary, but it may make code more readable

```
if condition1 and condition2:
        body1
elif condition1:
        body2
```

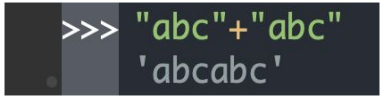
UNIVERSITY OF TORONTO

Review of Lecture

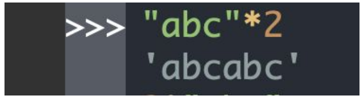
Topic 3: String Operations

Python String Operators: Concatenation and Repetition

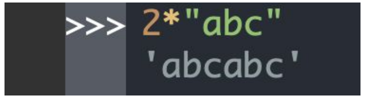
- Two useful operators that can be applied to strings: `*`, aka *repetition*, and `+`, aka *concatenation*
- Concatenation is a means for generating new values of type string. Note that the desired string can be achieved in multiple ways.
    - For example, let's generate "**abcabc**" in multiple ways



```
>>> "abc"+"abc"
'abcabc'
```



```
>>> "abc"*2
'abcabc'
```



```
>>> 2*"abc"
'abcabc'
```

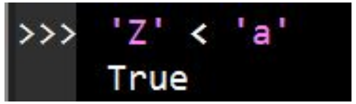
Summary:

Expression	Description	Example	Output
str1 + str2	concatenate str1 and str1	print('ab' + 'c')	abc
str1 * int1	concatenate int1 copies of str1	print('a' * 5)	aaaaa
int1 * str1	concatenate int1 copies of str1	print(4 * 'bc')	bcbcbcbc

Comparing Strings

- In Python, strings are compared based on the ASCII/UNICODE encoding of their components.
- The characters in both strings are compared one by one.
- When different characters are found then their ASCII/Unicode value is compared.
- The character with lower ASCII/Unicode value is considered to be smaller

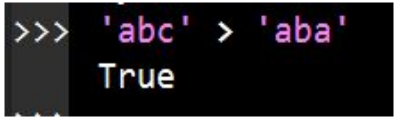
Examples:



```
>>> 'Z' < 'a'
True
```

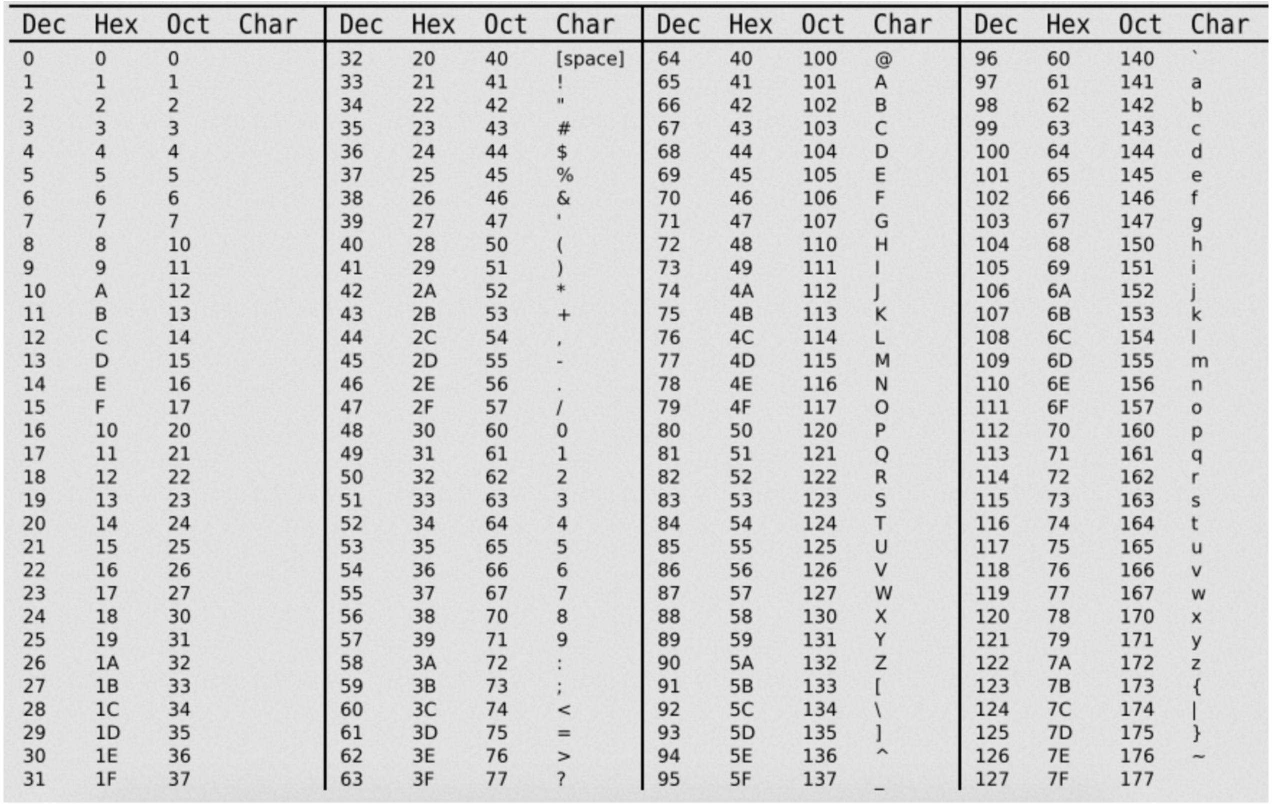
Z's encoding in ASCII is 90 while a's encoding is 97

c's encoding in ASCII is 99 while a's encoding is 97



```
>>> 'abc' > 'aba'
True
```

ASCII encoding



Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

String methods

Method name	Description
`isalnum()`	Returns True if string is alphanumeric
`isalpha()`	Returns True if string contains only alphabets
`isdigit()`	Returns True if string contains only digits
`isidentifier()`	Return True if string is valid identifier
`islower()`	Returns True if string is in lowercase
`isupper()`	Returns True if string is in uppercase
`isspace()`	Returns True if string contains only whitespace

A comprehensive list of string method can be found in the official documentation: https://docs.python.org/3/library/stdtypes.html#string-methods

APS106

UNIVERSITY OF TORONTO

Practice Problems

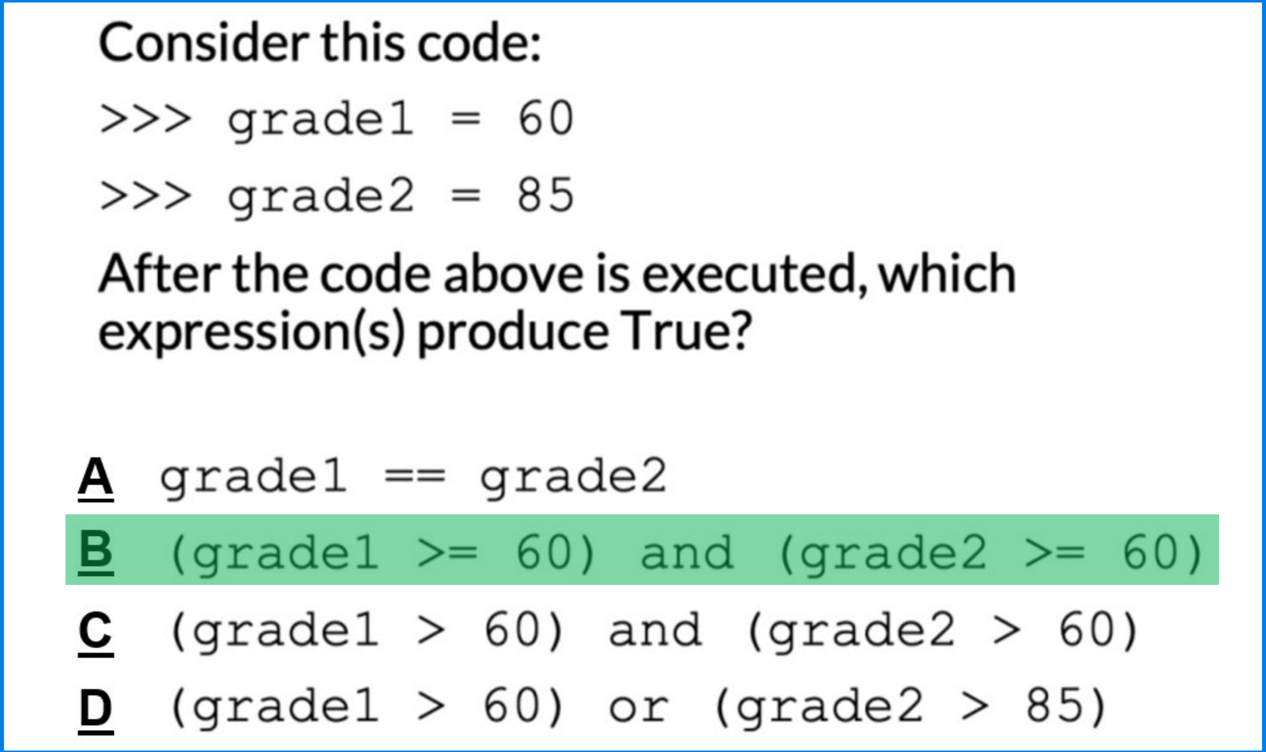
if nothing else, write #cleancode

slido

What is the answer in the image:

ⓘ Start presenting to display the poll results on this slide.

Review Practice Problem 1



Consider this code:

```
>>> grade1 = 60
>>> grade2 = 85
```

After the code above is executed, which expression(s) produce True?

A `grade1 == grade2`

[✓ **B** `(grade1 >= 60) and (grade2 >= 60)`]

C `(grade1 > 60) and (grade2 > 60)`

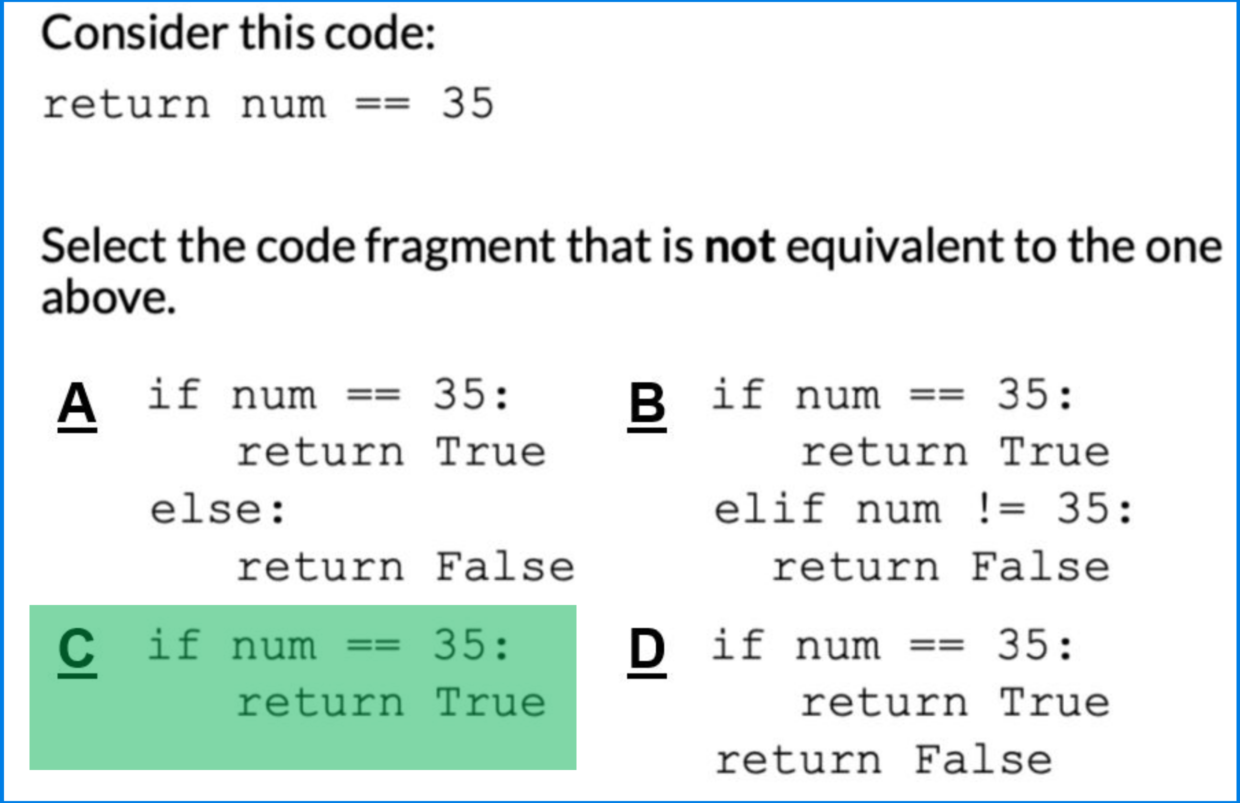
D `(grade1 > 60) or (grade2 > 85)`

slido

What is the answer in the image:

ⓘ Start presenting to display the poll results on this slide.

Review Practice Problem 2



Consider this code:

```
return num == 35
```

Select the code fragment that is **not** equivalent to the one above.

A

```
if num == 35:
    return True
else:
    return False
```

B

```
if num == 35:
    return True
elif num != 35:
    return False
```

[✓ **C**]

```
if num == 35:
    return True
```

D

```
if num == 35:
    return True
return False
```

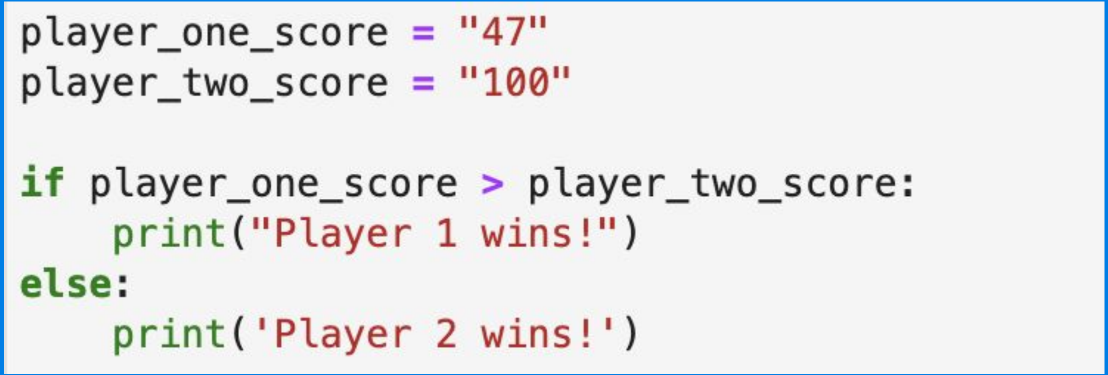
slido

What is the outcome of the following code?

ⓘ Start presenting to display the poll results on this slide.

Review Practice Problem 3

What is the outcome of the following code?



```python
player_one_score = "47"
player_two_score = "100"

if player_one_score > player_two_score:
    print("Player 1 wins!")
else:
    print('Player 2 wins!')
```

A. `Player 1 wins!` is printed
B. `Player 2 wins!` is printed
C. An error is thrown
D. None of the above

Coding Problem

Write a function that takes in 3 points, and returns the distance between the two points that are closest to each other.

Things to consider before you start writing your code:

1. What would be the parameters of your function?
   - `def closest_points(x1, y1, x2, y2, x3, y3)`
2. Maybe having one function just for finding distance may be useful?

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

slido

Any questions?

ⓘ Start presenting to display the poll results on this slide.