



Course Review (Boolean, branching)

Logical/ Boolean Operators

Logical operators **connect** Boolean values and expressions and **return** a Boolean value as a result.

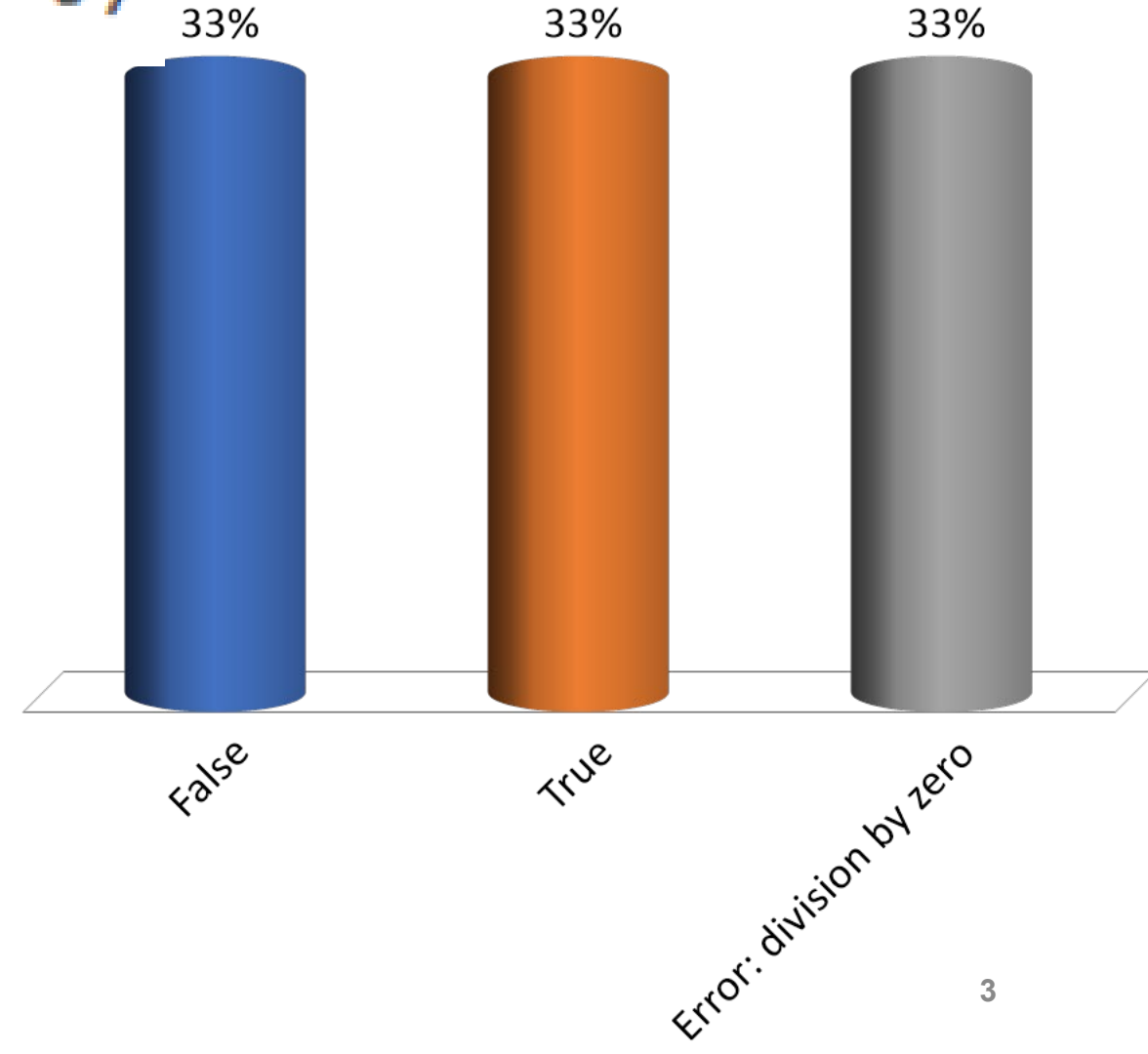
A	B	not A	A and B	A or B
False	False	True	False	False
False	True	True	False	True
True	False	False	False	True
True	True	False	True	True

Operator	Example	Meaning
not	not num < 0	Flip T/ F
and	(num1 > num2) and (num2 > num3)	Return True only if both are True
or	(num1 > num2) or (num2 > num3)	Return True if either one is True

What is the output of the following code?

```
print (3/0 > 3 and 3 < 0)
```

- A. False
- B. True
- ✓ C. Error: division by zero



What is the output of the following code?

```
| print(3 < 3 and 3/0 > 0)
```

- ✓ A. False
- B. True
- C. Error: division by zero

What is the output of the following code?

```
print(3 > 0 or 3/0 != 1)
```

- A. False
- ✓ B. True
- C. Error: division by zero



SHORT CIRCUIT EVALUATION

- Both the *and* and *or* operators perform short-circuit evaluation.
 - If the expression on the left side of the *and* operator is false, the expression on the right side will not be checked. Because the compound expression will be false if only one of the subexpressions is false, it would waste CPU time to check the remaining expression. So, when the *and* operator finds that the expression on its left is false, it short-circuits and does not evaluate the expression on its right.
 - If the expression on the left side of the *or* operator is true, the expression on the right side will not be checked. Because the compound expression will be true if only one of the subexpressions is true, it would waste CPU time to check the remaining expression. So, when the *or* operator finds that the expression on its left is true, it short-circuits and does not evaluate the expression on its right.
-

IF-ELSE

```
if condition:  
    indentedStatementBlockForTrueCondition  
else:  
    indentedStatementBlockForFalseCondition
```

(No *NEW* syntax)

Nested IF

```
if condition1:  
    if condition2:  
        SUITE A  
    else:  
        SUITE B  
else:  
    SUITE C
```

IF-ELIF-ELSE

```
if expression1:  
    suite1  
elif expression2:  
    suite2  
else:  
    suite3
```

**More on Selection
(Branching) SYNTAX**


What is the output of the code shown below?

```
if (7 < 0) and (0 < -9) :  
    print("Python")  
elif False or (100 > 0) :  
    print("good")  
else:  
    print("bad")
```

- A. Python
- ✓ B. good
- C. bad
- D. good bad
- E. None of above

What is the output of the code shown below?

```
x = 0
a = 3
b = -1
if a > 0:
    if b < 0:
        x = x + 5
    elif a > 5:
        x = x + 4
    else:
        x = x + 3
else:
    x = x + 2
print(x)
```

- A. 0
- B. 2
- C. 3
- D. 4
-  E. 5
- F. -1

A large, irregular orange watercolor splash or ink blot serves as the background for the title. It has a textured, painterly appearance with various shades of orange and some darker spots. The text is centered within this splash.

Advanced topics

1. CHAINED ASSIGNMENT

- We want more variables to take the same value:
- **a = b = 1**
- **a = b = c = 10**
- Don't make it too long or clumsy...
- Readability!!!
- 2. How about this? **a = b = 5 = c = 10**

WHAT IS THE OUTPUT OF THE FOLLOWING PYTHON CODE?

```
a = b = c = 10  
a = a - 5  
a = b  
b = a  
print (a + b + c)
```

- A. 25
- B. 20
- 😊 C. 30
- D. 10

SWAPPING TWO VALUES

- We want to swap values

a = 1

b = 2

- Can we do this?

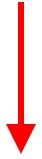
a = b

b = a

Computationally incorrect. Why?

SWAPPING TWO VALUES

- One standard way in many programming languages is to use a temporary variable as a buffer:



```
tmp = a
```

```
a = b
```

```
b = tmp
```

We can then swap the reference (computationally)

2. MULTIPLE ASSIGNMENT

- more than assign the result of a single expression to a single/multiple variables. (1:1, 1:n)
 - assigning multiple variables at one time.
- The left and right side must have the same number of elements. (n:n)
- Use comma for multiple assignment

a , b = 10, 20

- Note: It supports more than two elements

a , b , c = 10 , 11 , 12

- Make sure *same* number of elements on LHS and RHS

WHAT IS THE OUTPUT OF THE FOLLOWING PYTHON CODE?

```
a = 1  
b = 2  
a,b = b,a  
print (a, b)
```

- A. 1 2
- B. 1 1
- C. 2 2
- 😊 D. 2 1

Advanced topic: Formatting

Formatting Numbers

- Can format display of numbers on screen using built-in `format` function
 - Two arguments:
 - Numeric value to be formatted
 - Format specifier
 - Returns string containing formatted number
 - Format specifier typically includes precision and data type
 - Can be used to indicate scientific notation, comma separators, and the minimum field width used to display the value

The syntax of `format()` is:

```
format(value[, format_spec])
```

`format()` Parameters

The `format()` method takes two parameters:

- **value** - value that needs to be formatted
- **format_spec** - The specification on how the value should be formatted.

```
print(format(1.23456, '.2f'))
```




1.23

Formatting Numbers (cont'd.)

- The `%` symbol can be used in the format string of `format` function to format number as percentage


```
print(format(23/50, '%'))  
print(format(23/49, '%'))
```



```
46.000000%  
46.938776%
```

Specify 0 as the precision

```
print(format(23/50, '.0%'))  
print(format(23/49, '.0%'))
```



```
46%  
47%
```

Formatting Numbers (cont'd.)

- To format an integer using `format` function:
 - Use `d` as the type designator
 - Do not specify precision
 - Can still use `format` function to set field width or comma separator

```
print(format(123456, 'd'))  
print(format(123456, ',d'))  
print(format(123456, '10d'))  
print(format(123456, '10,d'))
```



```
123456  
123,456  
      123456  
      123,456
```

What is the output of the following code?

```
| print ("---", format(1.23456, "10.2%"))
```

- A. --- 1.23456%
- ✓ B. --- 123.46%
- C. --- 1.23
- D. --- 1.23456

Formatting Strings

- Python has awesome string formatters
- There are three methods to format a string in Python:
 - **%-format** - Python 2+
 - **{}** **str.format()** - Python 3.1+
 - **f-string** - Python 3.6+

Which of the following statement is the correct one to complete the code and generate the following output?

I am a year1 student

```
year = 1
```

```
I am a year 1 student
```

```
I am a year1student
```

```
I am a year1 student
```

- A. `print("I am a year", year, "student")`
- B. `print("I am a year", year, "student", sep="")`
- ✓ C. `print("I am a year"+str(year)+" student")`

The str.format() is a great method for string formatting:

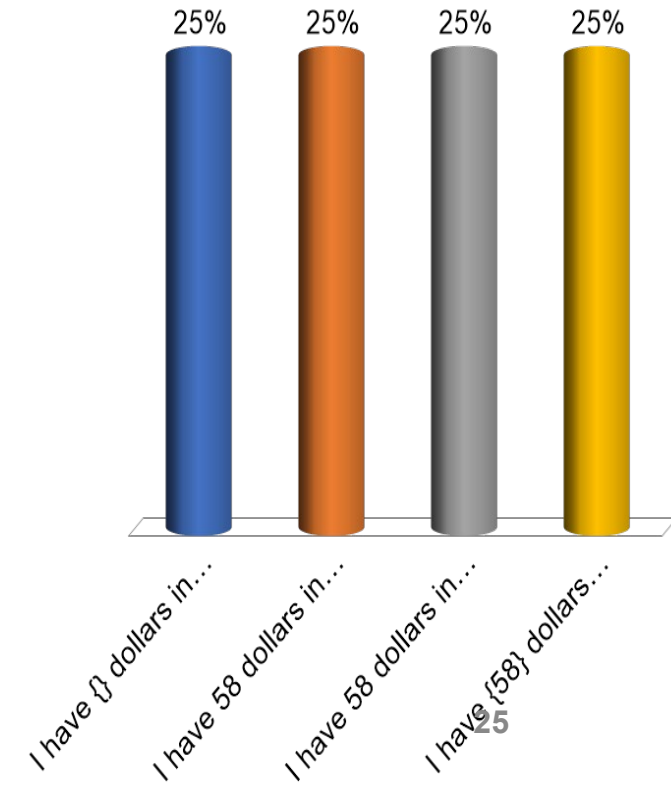
```
print("I am a year{} student".format(year))
```

Formatters work by putting in one or more replacement fields or placeholders — defined by a pair of curly braces {} — into a string and calling the str.format() method. You'll pass into the method the value you want to concatenate with the string. This value will be passed through in the same place that your placeholder is positioned when you run the program.

What is the output of the following code?

```
cash = 58
print("I have {} dollars in wallet! The book is {}".format(cash, 59.9))
```

- A. I have {} dollars in wallet! The book is {}.
- B. I have 58 dollars in wallet! The book is {}.
- ✓ C. I have 58 dollars in wallet! The book is 59.9.
- D. I have {58} dollars in wallet! The book is {59.9}.



```
print("Sammy is a {}, {}, and {} {}!".format("happy", "smiling", "blue", "shark"))
```

Output

```
Sammy is a happy, smiling and blue shark!
```

The string values contained in the tuple correspond to the following index numbers:

"happy"	"smiling"	"blue"	"shark"
0	1	2	3

Let's use the index numbers of the values to change the order that they appear in the string:

```
print("Sammy is a {3}, {2}, and {1} {0}!".format("happy", "smiling", "blue", "shark"))
```

Output

```
Sammy is a shark, blue, and smiling happy!
```

str.format() can avoid passing repeated arguments.

- You can use the same position or argument name in replacement fields if the same argument is used more than once in the string.

```
print("{0} is scary, but {0} programming language is interesting".format("Python"))
```

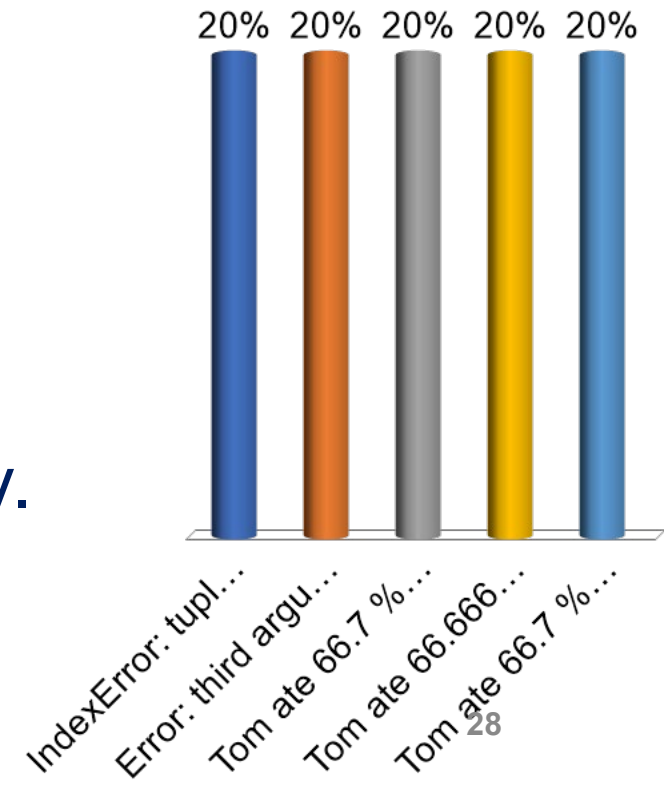


```
Python is scary, but Python programming language is interesting
```

What is the output of the following code?

```
print("{0} ate {1:.1f} % of the cake! {0} is so hungry.".format("Tom", (2/3)*100))
```

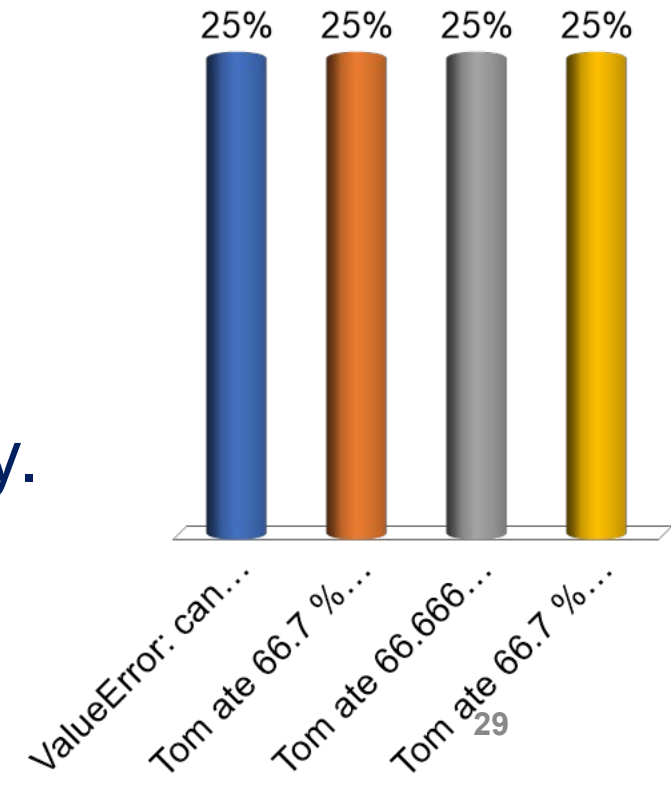
- A. IndexError: tuple index out of range
- B. Error: third argument is expected.
- C. Tom ate 66.7 % of the cake! {0} is so hungry.
- D. Tom ate 66.666667 % of the cake! Tom is so hungry.
- ✓ E. Tom ate 66.7 % of the cake! Tom is so hungry.



What is the output of the following code?

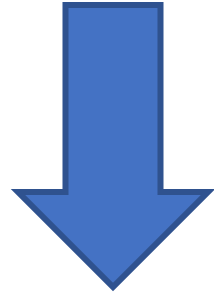
```
print("{} ate {:.1f} % of the cake! {0} is so hungry.".format("Tom", (2/3)*100))
```

- A. ValueError: cannot switch from automatic field numbering to manual field specification
- B. Tom ate 66.7 % of the cake! {0} is so hungry.
- C. Tom ate 66.666667 % of the cake! Tom is so hungry.
- D. Tom ate 66.7 % of the cake! Tom is so hungry.



You can pass one argument and access its multiple attributes or items in the string.

```
tp=('Turning point ID is:', 'CX1103')  
print('{v[0]} is {v[1]}'.format(v=tp))
```



Turning point ID is: is CX1103

Keyword arguments.

If you are creating a large formatting string, it is often much more readable and maintainable to use named replacement fields, so you don't have to keep counting out the arguments and figure out what argument goes where into the resulting string.

The formatting string can match against both positional *and* keyword arguments, and can use arguments multiple times:

```
"{1} {ham} {0} {foo} {1}".format(10, 20, foo='bar', ham='spam')
```



20 spam 10 bar 20

f-string

f-string, also called formatted string literal, is pretty much identical to `str.format()`, but just put arguments directly in the string. It has a unique function, though. You can call an argument's methods directly in the formatted string, while `str.format()` can call attributes or items only.

```
name = input("Please enter your name: ")  
print(f'{name.upper()}! Welcome to SC1003/CX1103')
```



```
Please enter your name: John  
JOHN! Welcome to SC1003/CX1103
```


Which of the following statement is the correct one to complete the code and generate the following output?







I am John, 21 years old, year1 Male student.

```
name, year, age, gender = "John", 1, 21, "Male"
```

- A. `print(f'I am {name}, {age} years old, year{year} {gender} student.')`
- B. `print('I am {}, {} years old, year{} {} student.'.format(name, age, year, gender))`
- C. `print('I am '+name+', '+str(age)+" years old, year"+str(year)+' '+gender+' student.')`
- D. None of the above
- ✓ E. All of the above



Some information about lab exercise

Color	RGB	Color
	<code>rgb(255,0,0)</code>	Red
	<code>rgb(0,255,0)</code>	Green
	<code>rgb(0,0,255)</code>	Blue
	<code>rgb(0,0,0)</code>	Black
	<code>rgb(128,128,128)</code>	Gray
	<code>rgb(255,255,255)</code>	White

Colors are displayed combining RED, GREEN, and BLUE light.

An RGB color value is specified with:
`rgb(RED , GREEN , BLUE)`.

Each parameter defines the intensity of the color as an integer between 0 and 255.

https://www.w3schools.com/colors/colors_rgb.asp