



# Course Review (Boolean, branching)

# Logical/ Boolean Operators

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

A	B	<b>not A</b>	<b>A and B</b>	<b>A or B</b>
False	False	True	False	False
False	True	True	False	True
True	False	False	False	True
True	True	False	True	True

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

slido



**What is the output of the following code?**

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

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

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

```
print(3/0 > 3 and 3 < 0)  
ZeroDivisionError: division by zero
```

slido



**What is the output of the following code?**

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

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

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

False

slido



**What is the output of the following code?**

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

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

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

True





# 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**

slido



**What is the output of the code ?**

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

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

good

slido



**What is the output of the code ?**

① Start presenting to display the poll results on this slide.

```
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)
```

5

A large, irregular orange watercolor splash or ink blot is centered on a white background. The splash has a textured, painterly appearance with various shades of orange and some darker spots. The text "Advanced topics" is written in white, sans-serif font, centered within the orange 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!!!



slido



**What is the output of the following python code?**

① Start presenting to display the poll results on this slide.

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

30

## 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

slido



**What is the output of the following python code?**

① Start presenting to display the poll results on this slide.

```
a,b = 1,2
a=b
b=a
print("First: ",a,b, end =" vs ")
a,b = 1,2
a,b = b,a
print("Second: ",a,b)
```

First: 2 2 vs Second: 2 1

# 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)

# Advanced topic: Formatting

		Built-in Functions		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	



# 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
```

slido



**What is the output of the following code?**

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

① Start presenting to display the poll results on this slide.

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

--- 123.46%

		Built-in Functions		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

# slido

Please fill up a line of statement in blank to complete the code to generate the following output.

Output:

please enter your year: 4

You are a year4 student

Hint code:

```
year = int(input("please enter your year: "))
```

---

```
year = int(input("please enter your year: "))  
print("You are a year"+str(year)+" student")
```

```
please enter your year: 4  
You are a year4 student
```



year = 1

---

- 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")`

I am a year 1 student

I am a year1student

I am a year1 student

# **Python Strings**

**Python has a built-in string class named "str" with many handy features**

# 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+

## Definition and Usage

The `format()` method formats the specified value(s) and insert them inside the string's placeholder.

The placeholder is defined using curly brackets: `{}`. Read more about the placeholders in the Placeholder section below.

The `format()` method returns the formatted string.

Syntax

```
string.format(value1, value2...)
```

The values can be of any data type.

```
year = int(input("please enter your year: "))  
print("You are a year"+str(year)+" student")  
print("You are a year{} student".format(year))
```

please enter your year: 4  
You are a year4 student  
You are a year4 student

- Formatters work by putting in one or more replacement fields or placeholders
- The placeholders can be identified using named indexes {price}, numbered indexes {0}, or even empty placeholders {}.
- 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.

```
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
```

# slido



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))
```

① Start presenting to display the poll results on this slide.



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

Tom ate 66.7 % of the cake! Tom is so hungry.

# slido



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

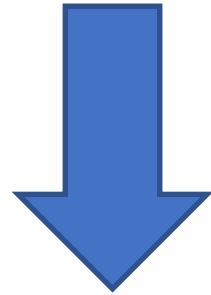
① Start presenting to display the poll results on this slide.

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

ValueError: cannot switch from automatic field numbering to manual field specification

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
```



# Some information about lab exercise

Color	RGB	Color
	rgb(255,0,0)	Red
	rgb(0,255,0)	Green
	rgb(0,0,255)	Blue
	rgb(0,0,0)	Black
	rgb(128,128,128)	Gray
	rgb(255,255,255)	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.