



# CL06 - Boolean Operators and Conditional Control Flow

# Announcements

Re: Quiz 00

- Median grade was 90% – great job!!
- Graded quizzes available on Gradescope
  - *Please review what you missed ASAP*; we will build on the topics covered in Quiz 00 throughout the course, and these foundational concepts are vital!
  - Don't understand a particular question/part of a memory diagram? Please come see us in Office Hours! Full list of hours on the site's [support page](#)
- *Regrade requests will be open till Thursday at 11:59pm*. Please submit a regrade request if you believe your quiz was not graded correctly according to the rubric

LS06 and LS07 (multiple choice questions) – due tonight at 11:59pm

[EX01 – Tea Party Planner](#) – due Saturday, May 24th at 11:59pm

Note: only boxed rubric items with check marks were applied to your quiz



#### Question 5

(no title)

4 / 4.5 pts

Output

✓ + 1 pt Output is 5.0 with no quotes OR the RV for the crunch function call frame

- 0.5 pts Extra lines of output in addition to 5.0 OR the RV for the crunch function call frame. (Only select if student got points for Q)

Stack: Globals

✓ + 0.5 pts crunch's id reference bound to a fn lines 3-5 in the heap (e.g. id:0)

+ 0.5 pts measure's id reference bound to a fn lines 8-10 in the heap (e.g. id:1)

Stack: crunch function call frame

✓ + 0.5 pts Frame is labeled "crunch" and has its own defined box/separation from the rest of the stack

✓ + 0.5 pts RA of 13

✓ + 0.5 pts RV of 5.0 (written as a float)

✓ + 0.5 pts a has a value of 6

✓ + 0.5 pts b has a value of 9

- 0.5 pts Extraneous frames on Stack (e.g. measure frame, or more than one crunch frame)

+ 0 pts Incorrect or blank

# Warm-up Questions

Given these two function definitions, reason through the questions below with your neighbors!

```
1  """Warmup question"""
2
3
4  def is_21(age: int) -> bool:
5      """Return whether age is at least 21."""
6      print("in is_21's function body")
7      return age == 21 or age > 21
8
9
10 def birthday(age: int) -> int:
11     """Increases age by 1."""
12     print("in birthday's function body")
13     return age + 1
```

- Which expression is valid, based on parameter and return type declarations?
  - `is_21(age=birthday(age=21))`
  - `birthday(age=is_21(age=21))`
- For the selected expression above, which function call expression evaluates first?
  - Inner-most function call based on parentheses
  - Outer-most function call based on parentheses
  - First function call encountered, reading from left to right, ignoring parentheses
- What is the *printed output* of evaluating the following? `is_21(age=21)`
- What is the *returned value* of evaluating the following? `is_21(age=21)`

# Relational Operators (Review)

These operators are placed between expressions of the same type\* to compare them.  
Relational operators evaluate to *boolean values*.

Operator Symbol	Verbalization	True Ex.	False Ex.
<code>==</code>	Is equal to?	<code>1 == 1</code>	<code>1 == 2</code>
<code>!=</code>	Is NOT equal to?	<code>1 != 2</code>	<code>1 != 1</code>
<code>&gt;</code>	Is greater than?	<code>1 &gt; 0</code>	<code>0 &gt; 1</code>
<code>&gt;=</code>	Is at least?	<code>1 &gt;= 0</code> or <code>1 &gt;= 1</code>	<code>0 &gt;= 1</code>
<code>&lt;</code>	Is less than?	<code>0 &lt; 1</code>	<code>1 &lt; 0</code>
<code>&lt;=</code>	Is at most?	<code>0 &lt;= 1</code> or <code>1 &lt;= 1</code>	<code>1 &lt;= 0</code>

\*Comparisons between int and float values will automatically convert (“type coerce”) the ints to floats.

# Relational Operator Practice

1.  $1 + 2 < 3 + 4$

Which operator must have higher precedence?  $<$  or  $+$ ?

2.  $110.0 \neq 110$

3.  $\text{"UNC"} == \text{"Unc"}$

4.  $\text{"UNC"} > \text{"DUKE"}$

Be careful using relational operators to compare strings!

- Python is a case-sensitive programming language (e.g.,  $\text{"U"} \neq \text{"u"}$ )
- Every character has a numerical ("ASCII") value associated with it. Strings are compared based on each character of the string's ASCII values, in order (Read an explanation [here](#).)

# Reasoning through the logical or operator

Recall the warm-up question...

```
4  def is_21(age: int) -> bool:
5      """Return whether age is at least 21."""
6      print("in is_21's function body")
7      return age == 21 or age > 21
```

`is_21` returns `True` if age is at least 21, and `False` otherwise. How must the `or` operator work?

How could we rewrite line 7 to simplify it using a different relational operator?

Expression	Evaluated Result
False or False	
True or False	
False or True	
True or True	

# Reasoning through the logical and operator

Consider the function...

```
16 def can_enter(age: int, has_id: bool) -> bool:
17     """Can you enter the 21+ event?"""
18     return age >= 21 and has_id
```

`can_enter` returns `True` if `age` is at least 21 and `has_id` is `True`, and `False` otherwise. How does the `and` operator work?

Expression	Evaluated Result
False and False	
True and False	
False and True	
True and True	

What must have higher precedence:  
`>=` (relational operator), or  
`and` (logical/boolean operator)?



# Reasoning through the logical not operator

Consider the function...

```
21 def can_eat(temp: int, allergic: bool) -> bool:
22     """Is it safe to eat this food?"""
23     return temp >= 165 and not allergic
```

`can_eat` returns `True` if `temp` is at least 165 and `allergic` is `False`, and `False` otherwise. How does the `not` operator work?

Expression	Evaluated Result
<code>not False</code>	
<code>not True</code>	

For this to be sensible, what must be the precedence of `not`, `and`, and `or`?

# Logical / Boolean Operators

Expression	Evaluation
False <b>or</b> False	False
True <b>or</b> False	True
False <b>or</b> True	True
True <b>or</b> True	True

Expression	Evaluation
False <b>and</b> False	False
True <b>and</b> False	False
False <b>and</b> True	False
True <b>and</b> True	True

Expression	Evaluation
<b>not</b> False	True
<b>not</b> True	False

## Precedence (highest to lowest):

0. Arithmetic operators (PEMDAS)
1. Relational Operators
2. Not
3. And
4. Or

# Conditionals

Control flow is *linear*

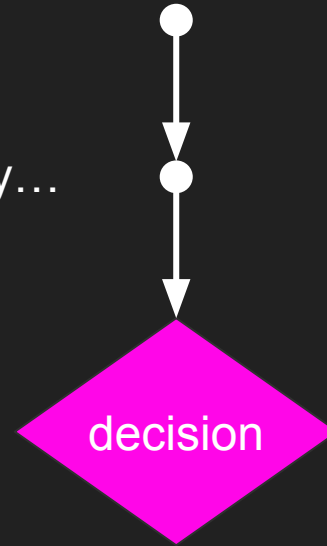
Going about your day...



Control flow is *linear*

Going about your day...

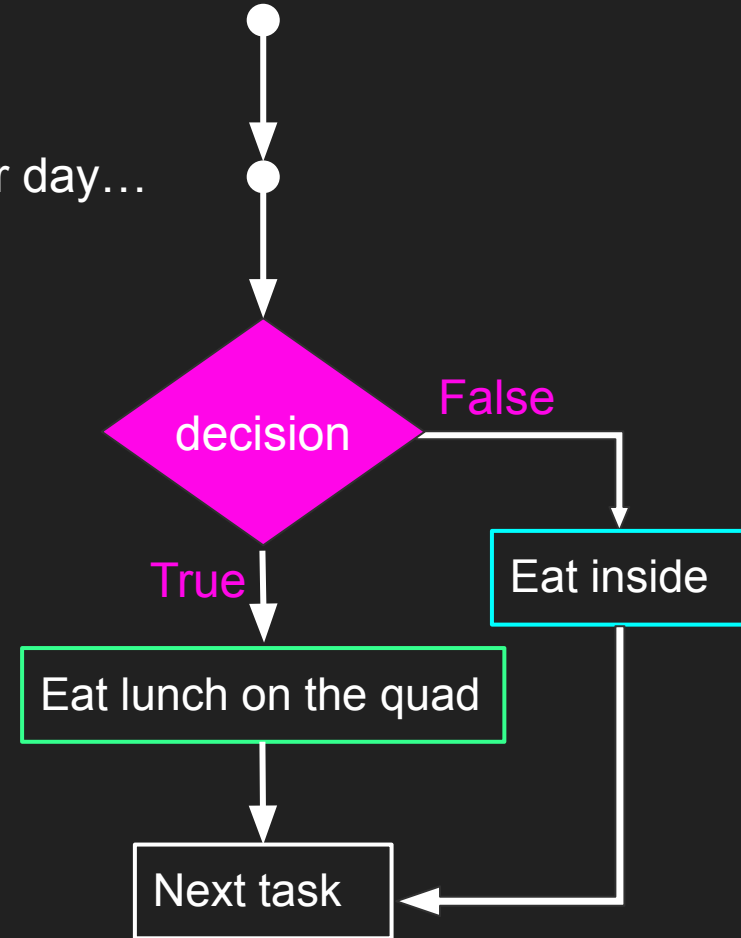
Is the weather nice?



# Control flow is *linear*

Going about your day...

Is the weather nice?



# If-then / *Conditional* Statements

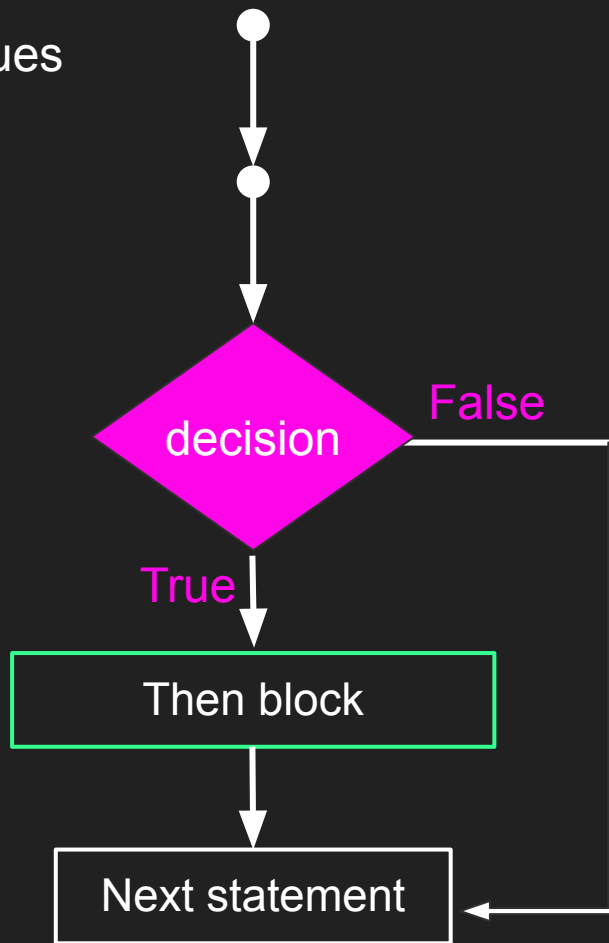
Code that behaves conditionally based on input values

`if <condition>:` ← `bool`

`<then, execute these statements>`

`<rest of program>`

```
def weather_info(temperature: int) -> None:
    """Function to make sense of the weather."""
    if temperature < 65:
        print("Don't forget a jacket!")
    print("Moving on...")
```



# If-then-else / *Conditional* Statements

Code that behaves conditionally based on input values

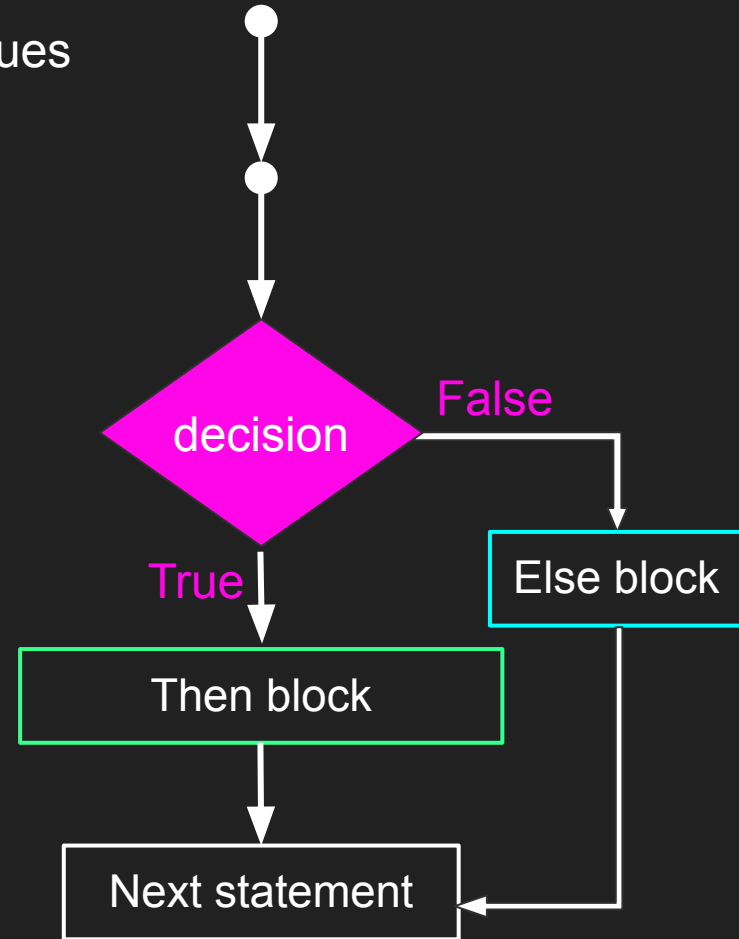
if <condition>:

<then, execute these statements>

else:

<execute these other statements>

<rest of program>





# If-then-else / *Conditional* Statements

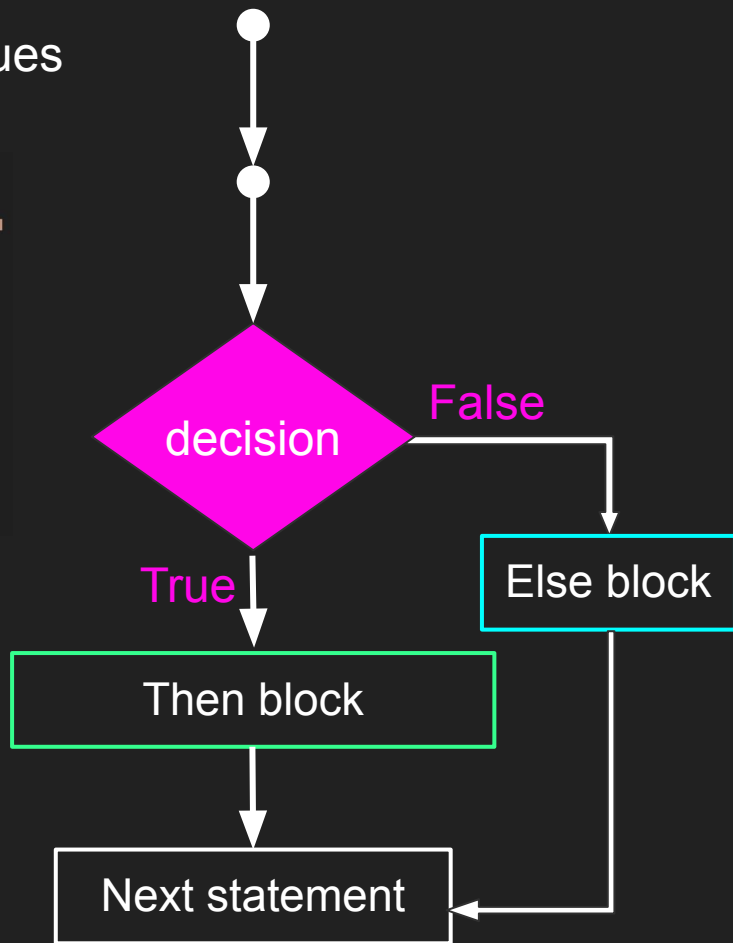
Code that behaves conditionally based on input values

```
4  def weather_info(temperature: int) -> None:
5      """Function to make sense of the weather."""
6      if temperature < 65:
7          print("Don't forget a jacket!")
8      else:
9          print("That's a comfortable temp!")
10     print("Moving on...")
```

Determine which lines would execute as a result of this function call: `weather_info(temperature=63)`

And this one:

`weather_info(temperature=75)`

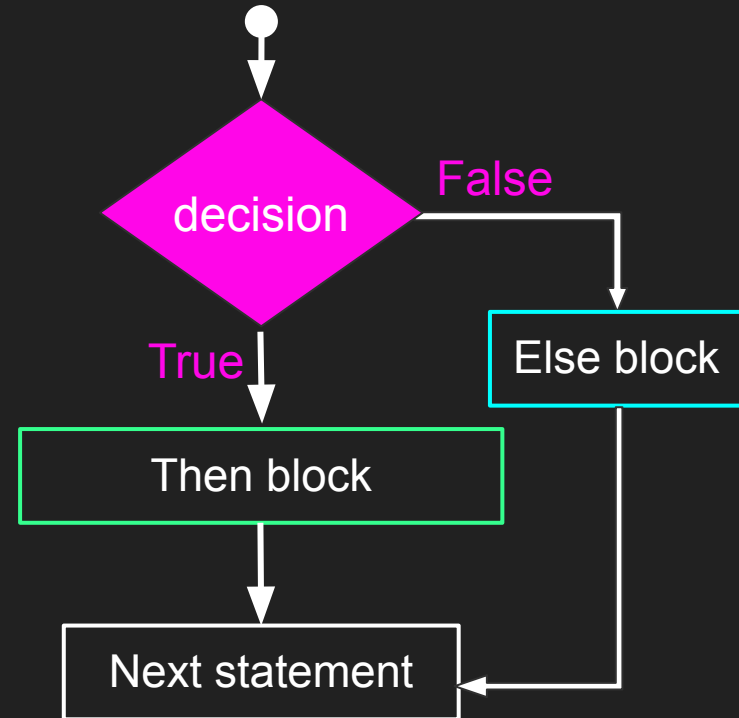


# General syntax and semantics

## Semantics:

1. When evaluation reaches an **if statement**, the **boolean test expression** is evaluated.
2. If the expression evaluates to **True**, control continues into the **then statement block**. If the then statement block completes without a return, control continues by moving on to the next statement after the if statement.
3. Otherwise, if the test expression evaluates to **False**, control *jumps over the then block* and continues to the next line, whether it is an **else statement block** (if there is one) or the next statement in the program.

**if** <condition>:  
    <then block>  
else:  
    <else block>  
<rest of program>



# If-elif-else / *Conditional* Statements

If you want to test multiple different conditions, you can use an “else if” (elif) statement!

**if** <condition>:

<then, execute these statements>

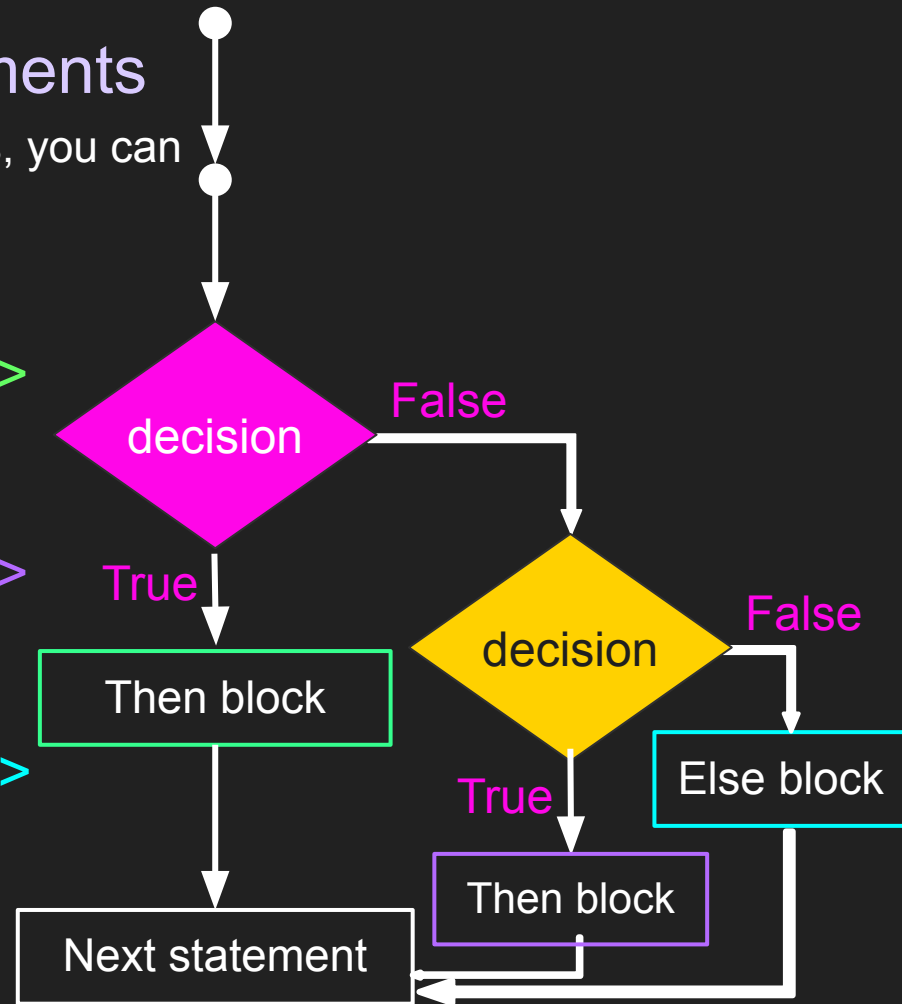
**elif** <different condition>:

<then, execute these statements>

**else:**

<execute these other statements>

<rest of program>



# Discussion

What is a decision you make in your day-to-day that you can express as an conditional (if-else) statement?

E.g. If I my assignment is due tomorrow, I start working on it. Else (it's not due tomorrow), I procrastinate another day.

*(This is bad behavior and I don't condone it!)*

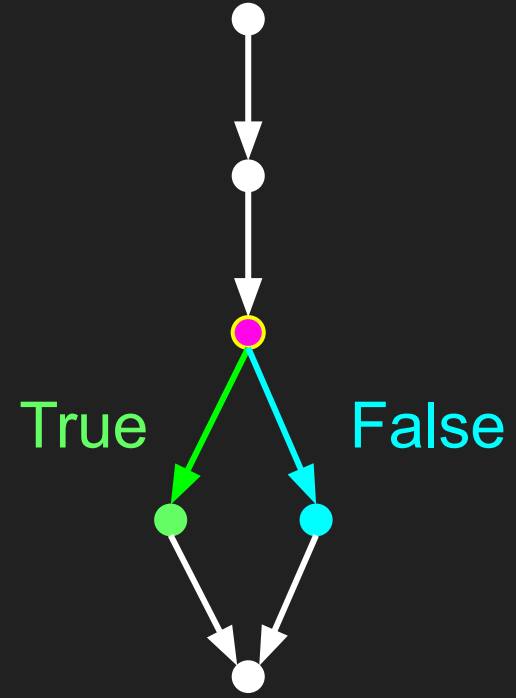
# Conditional Statements

if

:



else:



```
1  """Examples of conditionals."""
2
3
4  def number_report(x: int) -> None:
5      """Print some numerical properties of x"""
6      if x % 2 == 0:
7          print("Even")
8      else:
9          print("Odd")
10
11     if x % 3 == 0:
12         print("Divisible by 3")
13
14     if x == 0:
15         print("Zero")
16     else:
17         if x > 0:
18             print("Positive")
19         else:
20             print("Negative")
21
22     print("x is " + str(x))
23
24
25  number_report(x=110)
```

# Practice

Write a function called `check_first_letter` that takes as input two `strs`: `word` and `letter`

It should return `"match!"` if the first character of `word` is `letter`

Otherwise, it should return `"no match!"`

Examples:

- `check_first_letter(word="happy", letter="h")` would return `"match!"`
- `check_first_letter(word="happy", letter="s")` would return `"no match!"`

```
1  """Calling to and fro..."""
2
3
4  def ping(i: int) -> int:
5      print("ping: " + str(i))
6      if i <= 0:
7          return i
8      else:
9          return pong(i=i - 1)
10
11
12  def pong(i: int) -> int:
13      print("pong: " + str(i))
14      return ping(i=i - 1)
15
16
17  print(ping(i=2))
```



```
1  """Mysterious 'rev' from source (src) to destination (dest)!"""
2
3
4  def rev(src: str, i: int, dest: str) -> str:
5      """You happen upon a magical lil function..."""
6      if i >= len(src):
7          return dest
8      else:
9          return rev(src=src, i=i + 1, dest=src[i] + dest)
10
11
12  print(rev(src="lwo", i=0, dest=""))
```