



Boolean Operators and Conditional Control Flow

Announcements

Re: Quiz 00

- Graded quizzes will be available on Gradescope soon! Once they're released:
 - *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 visit us in Office Hours or Tutoring! Full list of hours on the site's [support page](#)
 - Please submit a regrade request *if you believe your quiz was not graded correctly according to the rubric*

LS05 and LS06 (multiple choice questions) – due tonight at 11:59pm

Note: You will be able to see the full rubric on Gradescope, but only boxed rubric items with check marks were applied to your quiz

**Question 5****(no title)****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

Reasoning through the logical or operator

Consider the function...

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

Expression	Evaluation
False and False	False
True and False	False
False and True	False
True and True	True

Expression	Evaluation
not False	True
not True	False

Precedence (highest to lowest):

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

Conditionals

Control flow is *linear*

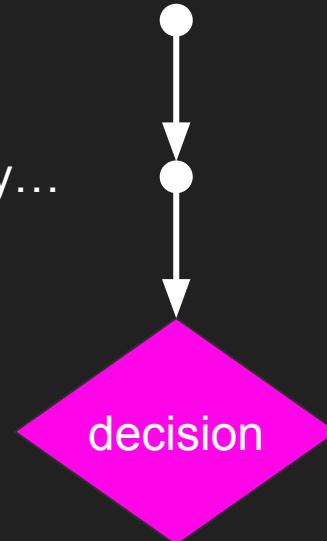
Going about your day...



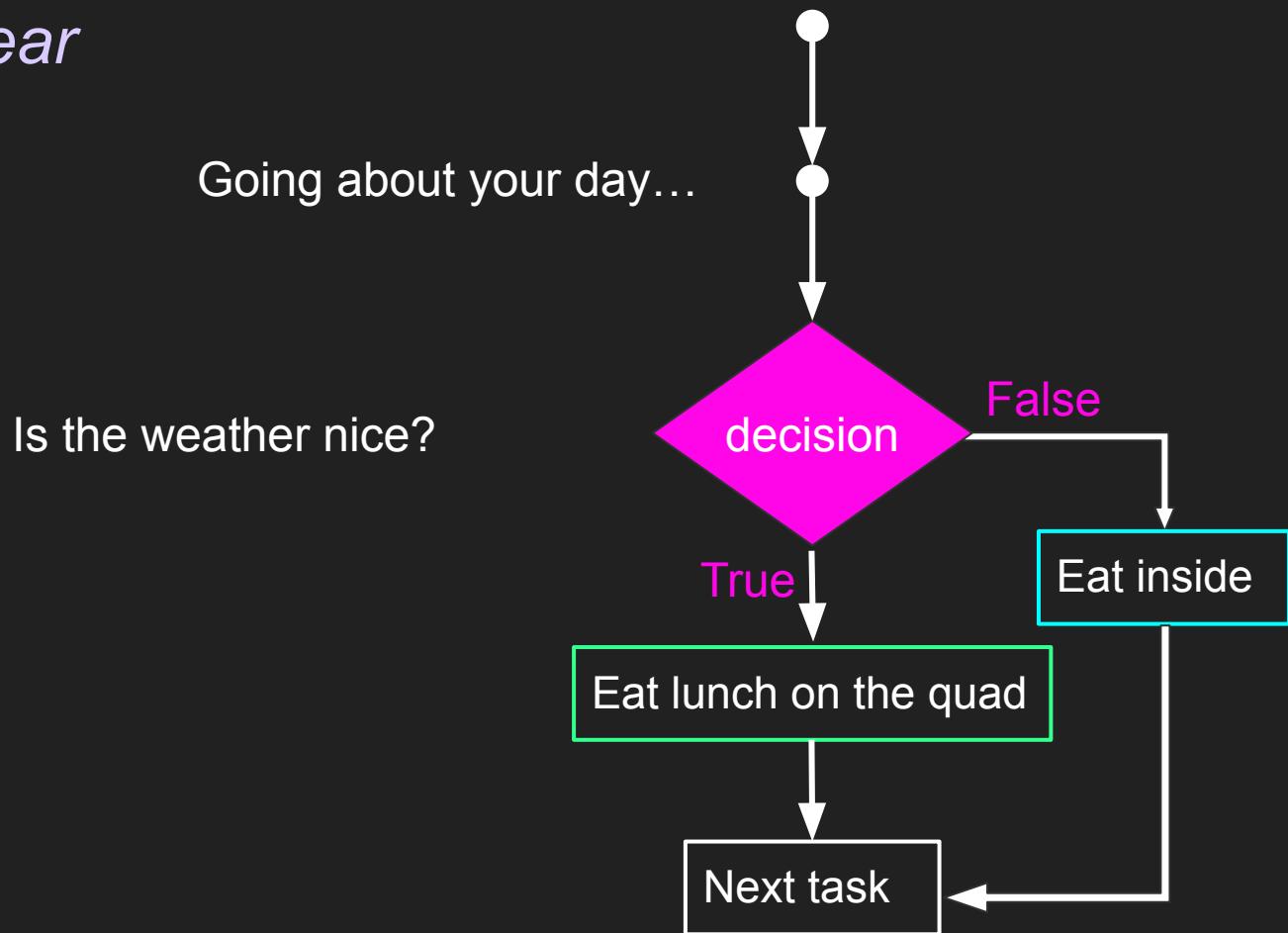
Control flow is *linear*

Is the weather nice?

Going about your day...



Control flow is *linear*

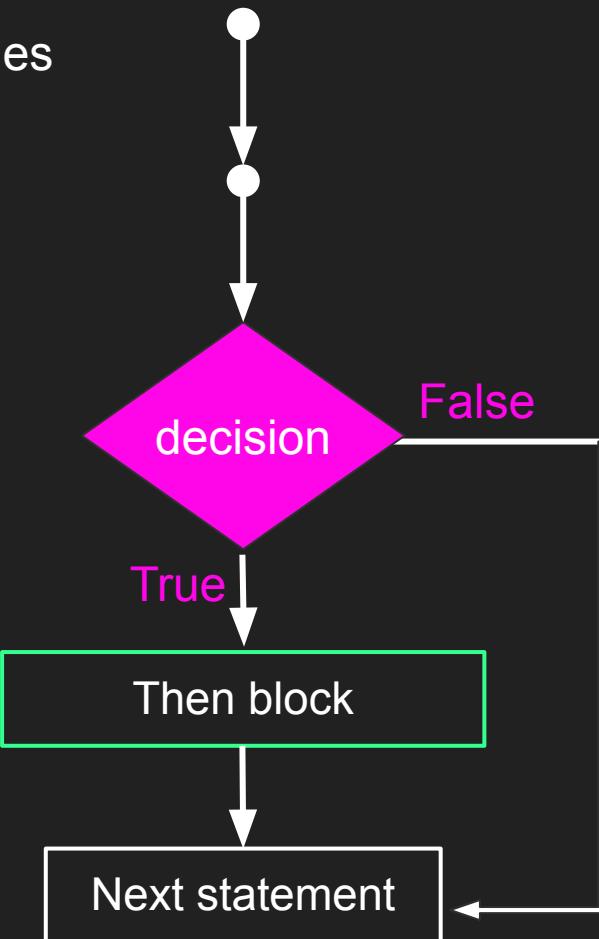


If-then / Conditional Statements

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

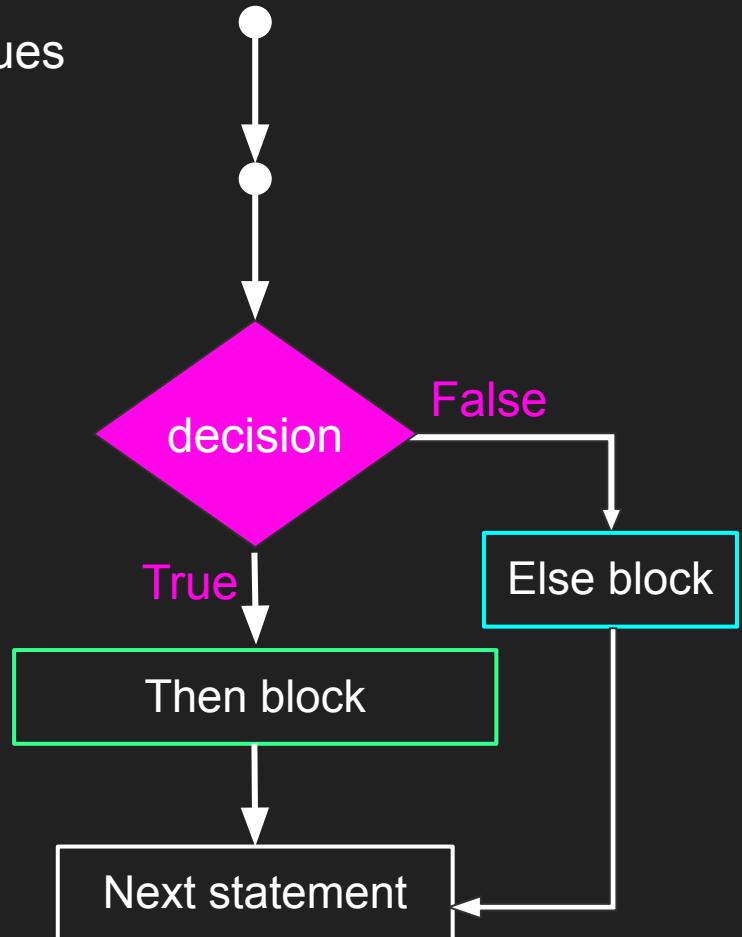
`if <condition>:`

`<then, execute these statements>`

`else:`

`<execute these other statements>`

`<rest of program>`



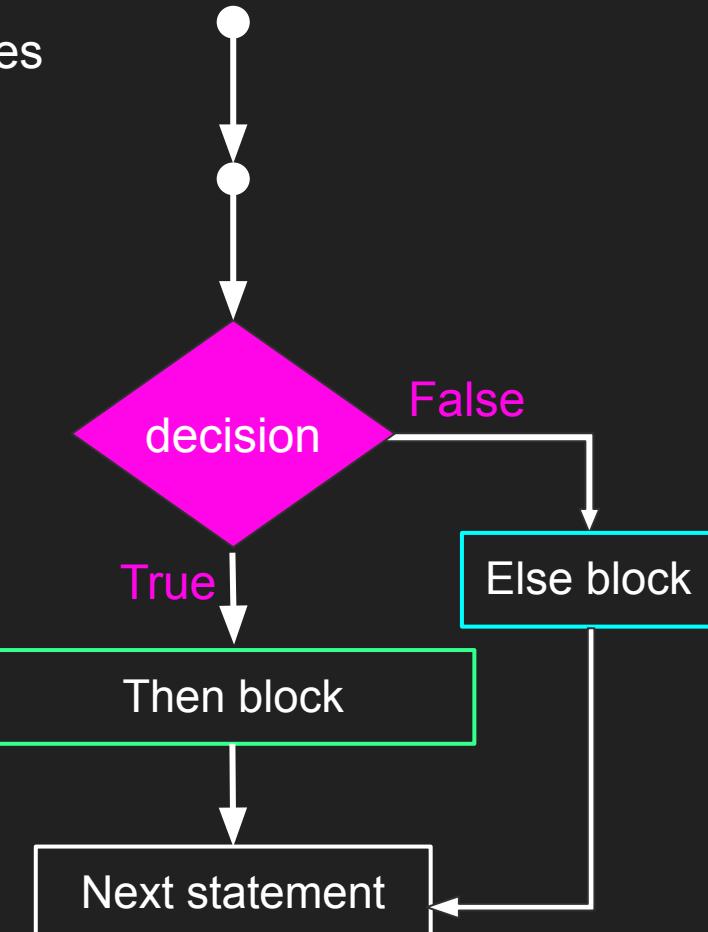
If-then-else / Conditional Statements

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)`

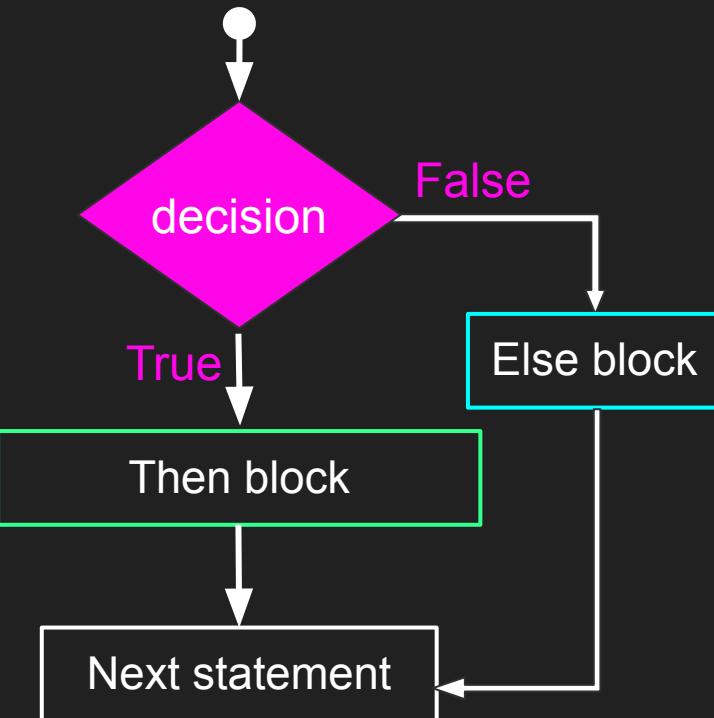


General syntax and semantics

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.

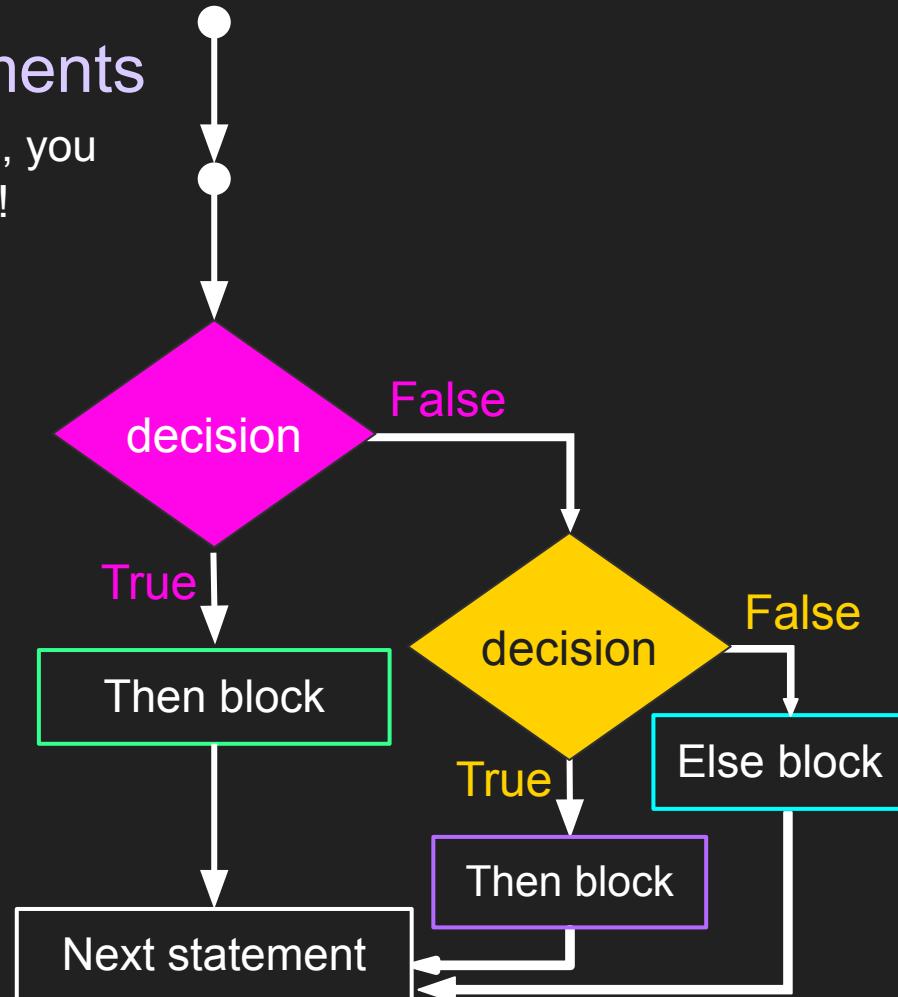
```
if <condition>:  
    <then block>  
else:  
    <else block>  
<rest of program>
```



If-elif-else / Conditional Statements

If you want to test multiple different conditions, you can use one or more “else if” (elif) statements!

```
if <condition>:  
    <then, execute these statements>  
  
elif <different condition>:  
    <then, execute these statements>  
  
else:  
    <execute these other statements>  
  
<rest of program>
```



Memory diagram

```
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)
```

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)
```

We could eliminate the need for a “nested” **if-then-else** statement (inside another conditional’s **else** statement) by adjusting this code to use an **elif** statement. How?

Practice

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

It should behave as follows:

- If `letter`'s value *doesn't* contain exactly one character, return "`letter`'s argument should be one character!"
- If the first character of `word` is the same as `letter`, return "match!"
- Otherwise, 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!"
- `check_first_letter(word="happy", letter="ha")` would return "`letter`'s argument should be one character!"

```
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=""))
```