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

→ I'll post the recording of Wednesday's lecture to the course site for anyone who was unable to come to class due to the icy conditions!

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) 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

Review: 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

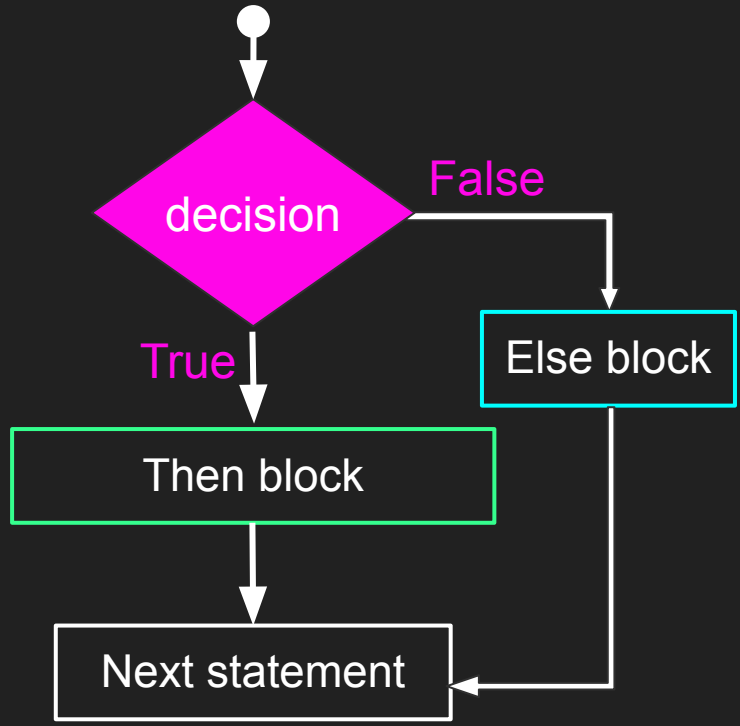
Mnemonic:
A
Real
Nerd
Always
Observes

General syntax and semantics

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

Semantics:

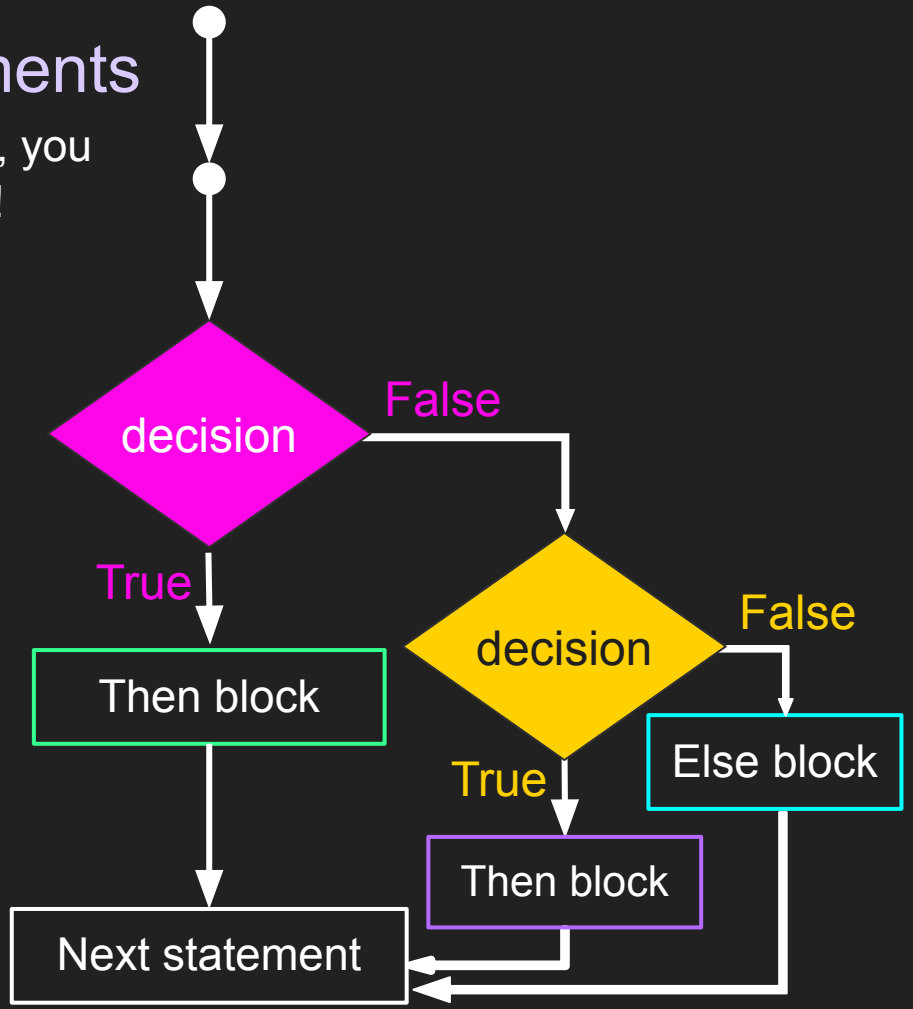
1. When evaluation reaches an **if statement**, the **boolean test expression** is evaluated.
2. If the expression evaluates to `True`, control continues into the **then statement block**. If the then statement block completes without a return, control continues by moving on to the next statement after 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-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

```
"""Examples of conditionals."""


def number_report(x: int) -> None:
    """Print some numerical properties of x"""
    if x % 2 == 0:
        print("Even")
    else:
        print("Odd")

    if x % 3 == 0:
        print("Divisible by 3")

    if x == 0:
        print("Zero")
    else:
        if x > 0:
            print("Positive")
        else:
            print("Negative")

    print("x is " + str(x))


number_report(x=110)
```

```python
"""Examples of conditionals."""


def number_report(x: int) -> None:
    """Print some numerical properties of x"""
    if x % 2 == 0:
        print("Even")
    else:
        print("Odd")

    if x % 3 == 0:
        print("Divisible by 3")

    if x == 0:
        print("Zero")
    else:
        if x > 0:
            print("Positive")
        else:
            print("Negative")

    print("x is " + str(x))


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 `str`s, 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!"`

```python
"""Calling to and fro..."""


def ping(i: int) -> int:
    print("ping: " + str(i))
    if i <= 0:
        return i
    else:
        return pong(i=i - 1)


def pong(i: int) -> int:
    print("pong: " + str(i))
    return ping(i=i - 1)


print(ping(i=2))
```

```python
"""Mysterious 'rev' from source (src) to destination (dest)!"""


def rev(src: str, i: int, dest: str) -> str:
    """You happen upon a magical lil function..."""
    if i >= len(src):
        return dest
    else:
        return rev(src=src, i=i + 1, dest=src[i] + dest)


print(rev(src="lwo", i=0, dest=""))
```