Conditional Control Flow

COMP110 - CL04

2024/01/30

Today is a Paper + Pencil or Tablet + Pencil day...
please keep laptops stowed away!

Illustration by Arthur Rackham, 1918, in *English Fairy Tales* by Flora Annie Steel

Announcements

- Quiz 0 - Median is approximately an 85%, will return by tomorrow
  - Please review and understand questions and diagram points missed immediately, these concepts will be important to have an understanding of on moving forward.
  - Not sure about a correct answer? Come see us in office hours for conceptual help! We *love* working through quiz-style conceptual problems. Please do not ask questions about content in regrade requests. The key is
- EX00 - Deadline was last night, but insurance points have you covered if you had trouble reaching full credit (see syllabus). Come to office hours if unable to hit 100%
- LS06 - Unicode, Emoji, Escape Sequences, and f-Strings Reading - Due Tomorrow
- ChatGPT, Google, and Explainability of Concepts Covered in COMP110

Today's Goals

- Warm-up Questions
- Relational Operators and Logical Operators
  - ==, != , >, >=, <, <=
- Conditional Control Flow with if-then-else Statements
- If time permits, function definitions that *call themselves!*

Warm-up Questions

Some misconceptions found on Thursday's quiz, and a new boolean idea!

```
"""Warmup question..."""


def is_21(age: int) -> bool:
    """Return whether age is at least 21."""
    print("in is_21")
    return age == 21 or age > 21


def age_next_year(age: int) -> int:
    """Calculates something's age next year."""
    print("in next_year")
    return age + 1
```

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

1. Which expression is valid based on parameter and return type declarations?
   a) `age_next_year(age=is_21(age=21))`
   b) `is_21(age=age_next_year(age=21))`
2. For the selected expression above, which function call expression *evaluates first*?
   a) Inner-most function call based on parens
   b) Outer-most function call based on parens
   c) First function call encountered, reading from left-to-right, ignoring parens
3. What is the *printed output* of evaluating: `is_21(age=21)`
4. What is the *returned value* of evaluating: `is_21(age=21)`

Relational Operators Refresher

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

English	Mathematical Notation	Python Operator
"is greater than"	>	>
"is at least"	≥	>=
"is less than"	<	<
"is at most"	≤	<=
"is equal to"	=	==
"is not equal to	≠	!=

** Comparisons made between int and float values will automatically convert ("type coerce") int values to floats*

Relational Operator Practice

On paper/tablet, write down each expression and its evaluated result.

A. `1 + 2 < 3 + 4`

Follow-on question: what operator ***must*** have higher precedence? < or +?

B. `"UNC" > "DUKE"`

Beware of string comparisons! The full reasoning for this is in the assigned lesson!

C. `110.0 != 110`

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")
7       return age == 21 or age > 21
```

1. is_21 returns True if age is at least 21 and false otherwise. How must the *or* operator work? Fill in the table below.

Expression	Evaluated Result (Fill In)
False **or** False	
True **or** False	
False **or** True	
True **or** True	

2. How would you rewrite line 7 to *simplify* it using a different relational operator?

Reasoning through the logical and operator

Consider this function...

```python
def can_enter(age: int, has_id: bool) -> bool:
    """Can you enter the 18+ club on Cottagecore night?"""
    return age >= 18 and has_id
```

1. Trusting this is a sensibly implemented function, reason through filling in the table.

Expression	Evaluated Result (Fill In)
False **and** False	
True **and** False	
False **and** True	
True **and** True	

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

Reasoning through the logical not operator

Consider this function...

```python
def can_enter(age: int, has_id: bool, looks_really_old: bool) -> bool:
    """Can you enter the 18+ club on Woodland Cottagecore night?"""
    return not age < 18 and has_id or looks_really_old
```

1. Trusting this is also a sensibly implemented function, reason through filling in the table.

Expression	Evaluated Result (Fill In)
not True	
not False	

2. For this to be sensible, what must be the precedence of not, 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 True	False
not False	True

Precedence (Highest-to-lowest):

0. **Arithmetic Operators (PEMDAS)**
1. **Relational Operators**
2. **not**
3. **and**
4. **or**

When in doubt, use parentheses!

For now, assume logical operators operate exclusively on boolean expression operands.

If-Then-Else / *Conditional* Statements

Writing code that behaves conditionally based on input values.

- Read the following function definition:

```
1    def taste_porridge(temp: int) -> str:
2        """Mmm bear porridge!"""
3        if temp == 140:
4            return "Just right!"
5        else:
6            if temp > 140:
7                return "Too hot!"
8            else:
9                return "Too cold!"
```

Predict 1) what is the evaluation of: `taste_porridge(temp=175)`

Predict 2) what is the evaluation of: `taste_porridge(temp=110)`

Predict 3) what is the evaluation of: `taste_porridge(temp=140)`

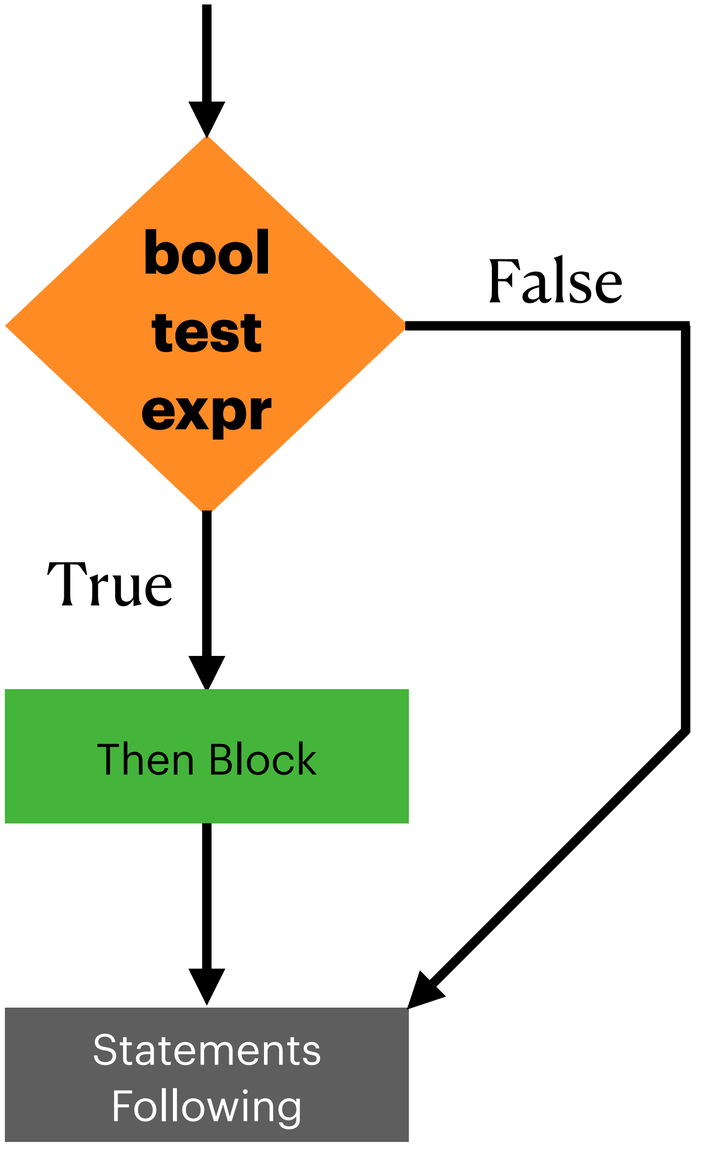
If-Then / *Conditional* Statements

General syntax and semantics

Syntax:

```
if bool_test_expression:
    ...then statements block...


...statements following if-else...
```



Semantics:

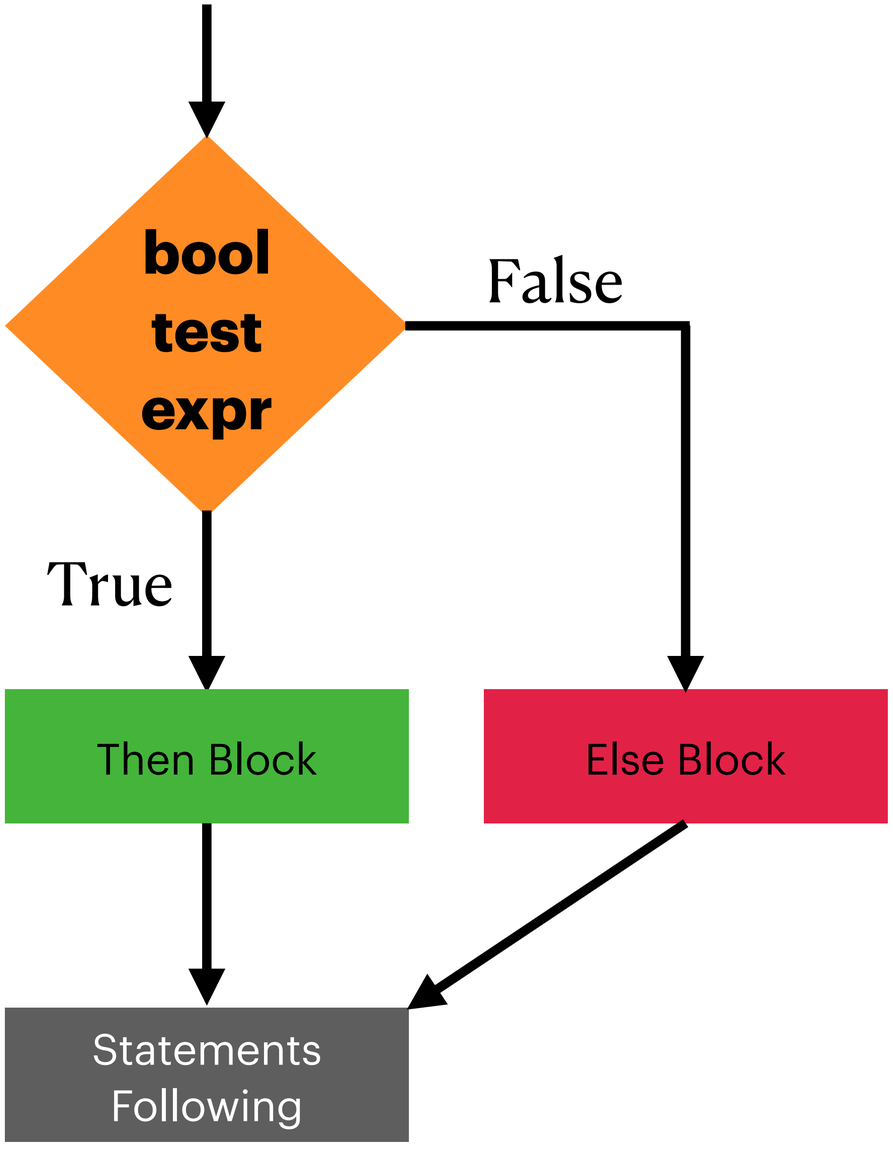
1. When evaluation reaches the *if* statement, the boolean test expression is evaluated.
2. If the test expression evaluates to true, control continues into the then statement block. If then block completes without a return, control continues by skipping else block.
3. Otherwise, if the test expression evaluates to false, control jumps over the then block and control continues to the next line.

If-Then-Else / *Conditional* Statements

General syntax and semantics

Syntax:

```
if bool_test_expression:
    ...then statements block...
else:
    ...else statements block...
...statements following if-else...
```



Semantics:

1. When evaluation reaches the *if* statement, the boolean test expression is evaluated.
2. If the test expression evaluates to true, control continues into the then statement block. If then block completes without a return, control continues by skipping else block.
3. Otherwise, if the test expression evaluates to false, control jumps over the then block and into the else block. If the else block completes without a return, control continues to the next line.

Diagram the Following Program

```
"""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)
```

Diagram the Following Program

```
"""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))
```

Diagram the Following Program

```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=""))
```

Submitting to Gradescope for Participation
From Hamilton 100

- Decide one of the two of you (or three...) to be the SUBMITTER

- From the SUBMITTER's cell phone:

1. Open CL assignment on Gradescope and make a submission

2. Upload a wide angle selfie-photo of your pair or group of 3 holding your *diagram* notes and giving a cozy thumbs up

3. Make your submission, **then add your partner(s) to your submission!!!**