CSC110 Lecture 5 Notes

Anatoly Zavyalov

September 22, 2020

1 Testing with `pytest`

1.1 Limitations of `doctest`

A function's doctest examples are primarily about communicating purpose, not verifying correctness.

1.2 Test terminology

A **unit test** is a block of code that checks for the correct behaviour of a function for one specific input.

A **test suite** is a collection of unit tests that checks the behavioour of a function of a (usually small) set of functions.

1.3 The `assert` statement

The `assert` statement evaluates an expression. If the expression is `True`, then nothing happens. If the expression is `False`, then an error occurs, signalling that the test failed.

2 Why Logic?

Mathematical logic is a **language of boolean expressions**. Here are some examples of statements:

- $3 + 2 = 5$
- Python's `sorted` function is correct
- My program is correct, assuming Python's `sorted` function is correct
- Python `set`s are more efficient than Python `list`s (for certain operations)

3 Propositional Logic (review from prep)

3.1 Terminology

A **proposition** is a statement that is either True or False.

Propositional operators:

- $\neg$ (NOT)
- $\vee$ (OR)
- $\wedge$ (AND)
- $\implies$ (implication/conditional)
- $\Leftrightarrow$ (bi-implication/biconditional)

3.2 Examples

Let $p =$ "Anatoly is cool" and $q =$ "Mario is cool".
$p \vee q$: "Anatoly is cool or mario is true" (Note that $\vee$ is inclusive.)
$p \Leftrightarrow q$: "Anatoly is cool if and only if Mario is cool"
$\neg p \implies q$: "If Anatoly is not cool, then Mario is cool".

3.3 More on implication

When we write down $p \implies q$: "If Anatoly is cool, then Mario is cool":

- p is the **hypothesis** ("Anatoly is cool")
- q is the **conclusion** ("Mario is cool")

p	q	$p \implies q$
F	F	T
F	T	T
T	F	F
T	T	T

Figure 1: Truth table of $p \implies q$

Here, we see that $p \implies q$ is **equivalent** to $\neg p \vee q$: "Anatoly is not cool or Mario is cool".

3.3.1 Contrapositive

$p \implies q$: "If Anatoly is cool, then Mario is cool".

This is **equivalent** to $\neg q \implies \neg p$: "If Mario is not cool, then Anatoly is not cool", which is the **contrapositive** of $p \implies q$.

3.3.2 Converse

$p \implies q$: "If Anatoly is cool, then Mario is cool".

This is **NOT equivalent** to $q \implies p$: "If Mario is cool then Anatoly is cool", which is the **converse** of $p \implies q$.

operator	notation	Python operation
NOT	$\neg p$	`not p`
AND	$p \wedge q$	`p and q`
OR	$\neg p$	`not p`
implication	$p \implies q$	`not p or q`
biconditional	$p \Leftrightarrow q$	`p == q`

4 Predicate logic

4.1 Terminology

A **predicate** is a function that returns a boolean.

Instead of "David is cool", "Mario is cool", "Anatoly is cool", etc., we can define a predicate:

$P(x)$: "x is cool", where x is a person.

4.2 The two quantifiers

$\forall x \in S, P(x)$ - "for all elements x of S, $P(x)$ is True"

$\exists x \in S, P(x)$ - "there exists an element x of S that makes $P(x)$ True"

4.2.1 Examples

Let S be teh set of all people and $IsCool$ the "x is cool" predicate defined over S.

$\forall p \in S, IsCool(P)$ - "Every person is cool"

$\exists p \in S, IsCool(p)$ - "There exists one person that is cool"

4.2.2 The quantifiers in Python

`all(bools)` : given a collection of booleans, return whether they are **all** `True`.

`any(bools)` : given a collection of booleans, return whether **at least one** is `True`.

5 Conditions and Filtering

5.1 Expressing conditions

"Every natural number n greater than 3 satisfies the inequality $n^2 + n \geq 20$."

Here are some attempts at translating this into logical symbols:

- $\forall n \in \mathbb{N}, n^2 + n \geq 20$ is **not** a correct translation.
- $\forall n \in \mathbb{N}, n > 3 \land n^2 + n \geq 20$ this is implying that every number is greater than 3, which is incorrect.
- $\forall n \in \mathbb{N}, n > 3 \implies n^2 + n \geq 20$ is correct.

5.2 The "forall-implies" template

"Every element of __ that __ satisfies __."

$\forall x \in S, P(x) \implies Q(x)$

5.3 Filtering in Python

In Python, we often want to **filter** a collection.

- Given a set of integers, find the sum of all the even numbers
- Given a set of (x, y) points, find the number of points within 1 unit of $(0, 0)$
- Given a list of strings, find all the strings that contain `'David'`

"greater than 3" is a **condition** that narrows the scope of the statement.

Normal (set) comprehension:

```
{<expression> for <variable> in <collection>}
```

Filtering comprehension:

```
{<expression> for <variable> in <collection> if <
    condition>}
```