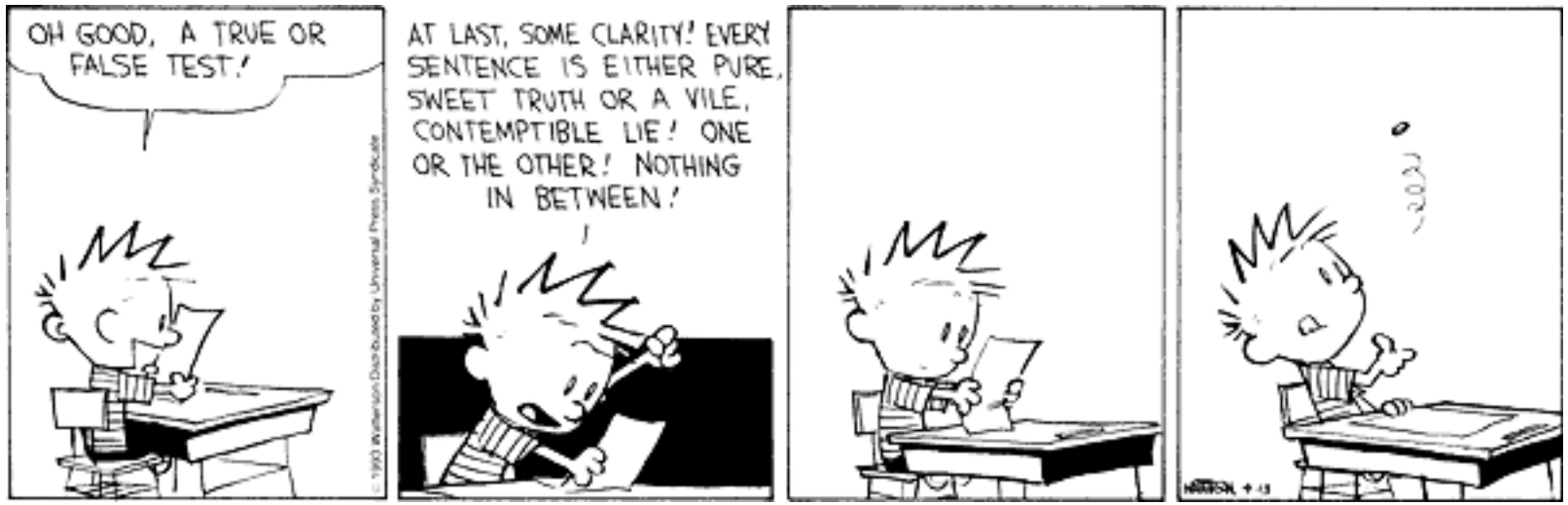
CSc 110, Autumn 2016

Lecture 14: Boolean Logic

Adapted from slides by Marty Stepp and Stuart Reges



Type `boolean`

- **`boolean`**: A logical type whose values are `True` and `False`.
  - A logical ***test*** is actually a `boolean` expression.
  - Like other types, it is legal to:
    - create a `boolean` variable
    - pass a `boolean` value as a parameter
    - return a `boolean` value from function
    - call a function that returns a `boolean` and use it as a test

```
minor     = age < 21
isProf    = "Prof" in name
lovesCSE = True

# allow only CS-loving students over 21
if (minor or isProf or not lovesCSE):
    print("Can't enter the club!")
```

Returning `boolean`

```
def is_prime(n):
    factors = 0;
    for i in range(1, n + 1):
        if (n % i == 0):
            factors += 1

    if (factors == 2):
        return True
    else:
        return False
```

- Calls to functions returning `boolean` can be used as tests:

```
if (is_prime(57)):
    ...
```

"Boolean Zen", part 1

- Students new to `boolean` often test if a result is `True`:

```
if (isPrime(57) == True):     # bad
    ...
```

- But this is unnecessary and redundant. Preferred:

```
if (isPrime(57)):             # good
    ...
```

- A similar pattern can be used for a `False` test:

```
if (isPrime(57) == False):    # bad
if (not isPrime(57)):         # good
```

"Boolean Zen", part 2

- Functions that return `boolean` often have an `if/else` that returns `True` or `False`:

```
def both_odd(n1, n2):
    if (n1 % 2 != 0 and n2 % 2 != 0):
        return True
    else:
        return False
```

  - But the code above is unnecessarily verbose.

Solution w/ `boolean` variable

- We could store the result of the logical test.

```
def bothOdd(n1, n2):
    test = (n1 % 2 != 0 and n2 % 2 != 0)
    if (test):    # test == True
        return True
    else:         # test == False
        return False
```

- Notice: Whatever `test` is, we want to return that.
  - If `test` is `True`, we want to return `True`.
  - If `test` is `False`, we want to return `False`.

Solution w/ "Boolean Zen"

- Observation: The `if/else` is unnecessary.
  - The variable `test` stores a `boolean` value;
    its value is exactly what you want to return. So return that!

```
def both_odd(n1, n2):
    test = (n1 % 2 != 0 and n2 % 2 != 0)
    return test
```

- An even shorter version:
  - We don't even need the variable `test`.
    We can just perform the test and return its result in one step.

```
def both_odd(n1, n2):
    return (n1 % 2 != 0 and n2 % 2 != 0)
```

"Boolean Zen" template

- Replace

```
def name(parameters):
    if (test):
        return True
    else:
        return False
```

- with

```
def name(parameters):
    return test
```

Improve the `is_prime` function

- How can we fix this code?

```
def is_prime(n):
    factors = 0;
    for i in range(1, n + 1):
        if (n % i == 0):
            factors += 1

    if (factors != 2):
        return False
    else:
        return True
```

De Morgan's Law

- **De Morgan's Law**: Rules used to negate boolean tests.
  - Useful when you want the opposite of an existing test.

Original Expression	Negated Expression	Alternative
`a and b`	`not a or not b`	`not(a and b)`
`a or b`	`not a and not b`	`not(a or b)`

  - Example:

Original Code	Negated Code
`if (x == 7 and y > 3):` `  ...`	`if (x != 7 or y <= 3):` `  ...`

Boolean practice questions

- Write a function named `is_vowel` that returns whether a `String` is a vowel (a, e, i, o, or u), case-insensitively.
  - `is_vowel("q")` returns `False`
  - `is_vowel("A")` returns `True`
  - `is_vowel("e")` returns `True`

- Change the above function into an `is_non_vowel` that returns whether a `String` is any character except a vowel.
  - `is_non_vowel("q")` returns `True`
  - `is_non_vowel("A")` returns `False`
  - `is_non_vowel("e")` returns `False`

Boolean practice answers

```
# Enlightened version.  I have seen the true way (and false way)
def is_vowel(s):
    return s == 'a' or s == 'A' or s == 'e' or s == 'E' or s =='i' or s == 'I'
           or s == 'o' or s == 'O' or s == 'u' or s =='U'


# Enlightened "Boolean Zen" version
def is_non_vowel(s):
    return not(s == 'a') and not(s == 'A') and not(s == 'e') and not(s == 'E')
           and not(s =='i') and not(s == 'I') and not(s == 'o') and
           not(s == 'O') and not(s == 'u') and not(s =='U')

    # or, return not is_vowel(s)
```

When to return?

- Functions with loops and return values can be tricky.
  - When and where should the function return its result?

- Write a function `seven` that uses `randint` to draw up to ten lotto numbers from 1-30.
  - If any of the numbers is a lucky 7, the function should stop and return `True`. If none of the ten are 7 it should return `False`.
  - The method should print each number as it is drawn.

```
15 29 18 29 11 3 30 17 19 22 (first call)
29 5 29 4 7                  (second call)
```

Flawed solution

```
# Draws 10 lotto numbers; returns True if one is 7.
def seven():
    for i in range(1, 11):
        num = randint(1, 30)
        print(str(num) + " ", end='')

        if (num == 7):
            return True;
        else:
            return False;
```

- The function always returns immediately after the first draw.
- This is wrong if that draw isn't a 7; we need to keep drawing.

Returning at the right time

```
# Draws 10 lotto numbers; returns True if one is 7.
def seven():
    for i in range(1, 11):
        num = randint(1, 30)
        print(str(num) + " ", end='')

        if (num == 7):    # found lucky 7; can exit now
            return True

    return False   # if we get here, there was no 7
```

- Returns `True` immediately if 7 is found.
- If 7 isn't found, the loop continues drawing lotto numbers.
- If all ten aren't 7, the loop ends and we return `False`.

`while` loop question

- Write a function `digit_sum` that accepts an integer parameter and returns the sum of its digits.
  - Assume that the number is non-negative.
  - Example: `digit_sum(29107)` returns 2+9+1+0+7 or `19`
  - Hint: Use the `%` operator to extract a digit from a number.

`while` loop answer

```
def digit_sum(n):
    n = abs(n)                          # handle negatives

    sum = 0
    while (n > 0):
        sum = sum + (n % 10)            # add last digit
        n = n // 10                     # remove last digit

    return sum
```

Boolean return questions

- `has_an_odd_digit` : returns `True` if <u>any</u> digit of an integer is odd.
  - `has_an_odd_digit(4822116)` returns `True`
  - `has_an_odd_digit(2448)` returns `False`

- `all_digits_odd` : returns `True` if <u>every</u> digit of an integer is odd.
  - `all_digits_odd(135319)` returns `True`
  - `all_digits_odd(9174529)` returns `False`

- `is_all_vowels` : returns `True` if <u>every</u> char in a `String` is a vowel.
  - `is_all_vowels("eIeIo")` returns `True`
  - `is_all_vowels("oink")` returns `False`

Boolean return answers

```
def has_an_odd_digit(n):
    while (n != 0):
        if (n % 2 != 0):    # check whether last digit is odd
            return True
        n = n // 10
    return False

def all_digits_odd(n):
    while (n != 0) :
        if (n % 2 == 0):    # check whether last digit is even
            return False
        n = n // 10
    return True


def is_all_vowels(s):
    for i in range(0, len(s)):
        letter = s[i, i + 1]
        if (not is_vowel(letter)):
            return false
    return true
```