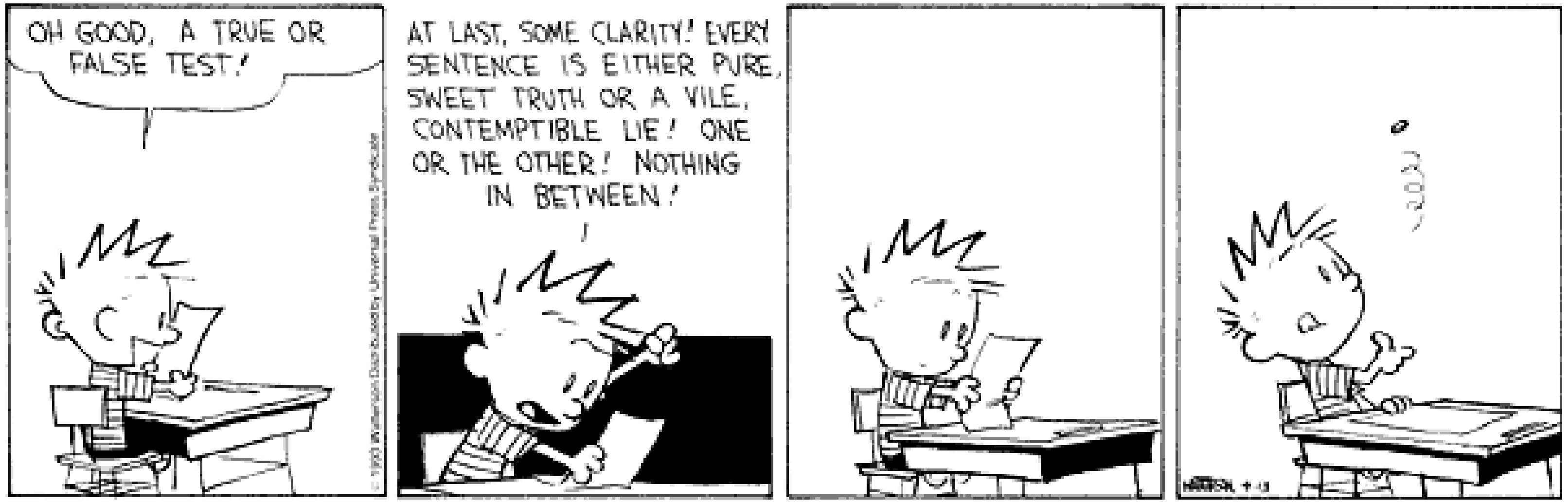
CSc 110, Spring 2017

Lecture 13: Random numbers and Boolean Logic

Adapted from slides by Marty Stepp and Stuart Reges


OH GOOD, A TRUE OR FALSE TEST!
AT LAST, SOME CLARITY! EVERY SENTENCE IS EITHER PURE, SWEET TRUTH OR A VILE, CONTEMPTIBLE LIE! ONE OR THE OTHER! NOTHING IN BETWEEN!

Random question

- Write a program that plays an adding game.
  - Ask user to solve random adding problems with 2-5 numbers.
  - The numbers to add are between 1 and 10
  - The user gets 1 point for a correct answer, 0 for incorrect.
  - The program stops after 3 incorrect answers.

```
4 + 10 + 3 + 10 = 27
9 + 2 = 11
8 + 6 + 7 + 9 = 25
Wrong! The answer was 30
5 + 9 = 13
Wrong! The answer was 14
4 + 9 + 9 = 22
3 + 1 + 7 + 2 = 13
4 + 2 + 10 + 9 + 7 = 42
Wrong! The answer was 32
You earned 4 total points
```

Random answer - main

```
# Asks the user to do adding problems and scores them.
from random import *

def main():
    # play until user gets 3 wrong
    points = 0
    wrong = 0
    while (wrong < 3):
        result = play()      # play one game
        if (result == 0):
            wrong += 1
        else:
            points += 1

    print("You earned " + str(points) + " total points.")
```

Random answer - play

```
# Builds one addition problem and presents it to the user.
# Returns 1 point if you get it right, 0 if wrong.
def play():
    # print the operands being added, and sum them
    num_operands = randint(2, 5)
    sum = randint(1, 10)
    print(sum, end='')

    for i in range(2, num_operands + 1):
        n = randint(1, 10)
        sum = sum + n
        print(" + " + str(n), end='')
    print(" = ", end='')

    # read user's guess and report whether it was correct
    guess = input()
    if (guess == sum):
        return 1
    else:
        print("Wrong! The answer was " + str(total))
        return 0
```

Type `bool` (Review)

- **`boolean`**: A logical type with only two values `True` and `False`.
  - A logical ***test*** is an expression of type `bool`.
  - As with other types, it is legal to:
    - assign a `bool` value to a variable
    - pass a `bool` value as a parameter
    - return a `bool` value from function
    - call a function that returns a `bool` value and use it as a test

```
minor    = age < 21
isProf   = name.startswith("Prof")
lovesCSE = True

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

Using booleans

- Why is type `bool` useful?
    - Can capture a complex logical test result and use it later
    - Can write a function that does a complex test and returns it
    - Makes code more readable
    - Can pass around the result of a logical test (as param/return)

```
low_Sodium    = sodium >= 35 and sodium < 140
low_Sugar     = sugar >= 5 and sugar < 12
vitamin_C     = c_count >= 100 and c_count <= 350
if ((low_Sodium and low_Sugar) or vitamin_Rich):
    print("Enjoy your healthy snack!")
else:
    print("Eat your snack in moderation.")
```

Returning booleans

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

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

- Calls to functions returning booleans can be used as tests:

```
if (is_prime(x)):
    ...
```

"Boolean Zen", part 1

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

```
if (is_prime(x) == True):      # bad
    ...
```

- But this is redundant. Preferred:

```
if (is_prime(x)):              # good
    ...
```

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

```
if (is_prime(x) == False):     # bad
if (not is_prime(x)):          # good
```

"Boolean Zen", part 2

- Functions that return booleans 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
```

  - Can this be shortened and improved?

Solution w/ variable assignment

- We could store the result of the logical test.

```
def both_odd(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` is assigned a value of type `bool`; 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 = factors + 1

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

Logic Question

- Consider the statement:
  - It is not true that he took Art History and Physics 101
- Is this an equivalent statement?
  - He did not take Art History or he did not take Physics 101

De Morgan's Laws

- **De Morgan's Laws**: Rules used to negate boolean tests involving and and or.
  - 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 not(x == 7 and y > 3):`
	`if (x =! 7 or y <= 3):`

Boolean practice questions

- Write a function `is_vowel(c)` that returns `True` if the 1 character string `c` is a vowel (a, e, i, o, or u) or `False` otherwise. Ignore case.
  - `is_vowel("q")` returns `False`
  - `is_vowel("A")` returns `True`
  - `is_vowel("e")` returns `True`

- Change the above function into `is_non_vowel(c)` that returns `True` if `c` is any character except a vowel and `False` otherwise.
  - `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(c):
    c = c.lower()       # allows testing for only lower case
    return c == 'a' or c == 'e' or c =='i' or c == 'o' or c == 'u'


# Enlightened "Boolean Zen" version
def is_non_vowel(c):
    c = c.lower()
    return not(c == 'a' or c == 'e' or c =='i' or c == 'o' or c == 'u')

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

When to return?

- Consider a function with a loop and a return value:
  - 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 immediately return `True`. If none of the ten are 7 it should return `False`.
  - The function 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.
- If the draw isn't a 7, we need to keep drawing (up to 10 times).

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`.

Sidebar…

- Write a function `digit_sum(n)` that accepts an integer parameter and returns the sum of its digits.
  - Assume that the number is non-negative.
  - Example: `digit_sum(29107)` returns `19`

    (19 is the sum of 2+9+1+0+7)

  - Hint: Use the `%` operator to extract a digit from a number.
  - Hint: Use the // operator to remove the last digit

Summing digits answer

```
def digit_sum(n):

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

    return sum
```

Boolean return questions

- `has_an_odd_digit` : returns `True` if any 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 every digit of an integer is odd.
  - `all_digits_odd(135319)` returns `True`
  - `all_digits_odd(9174529)` returns `False`

- `is_all_vowels` : returns `True` if every 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
```