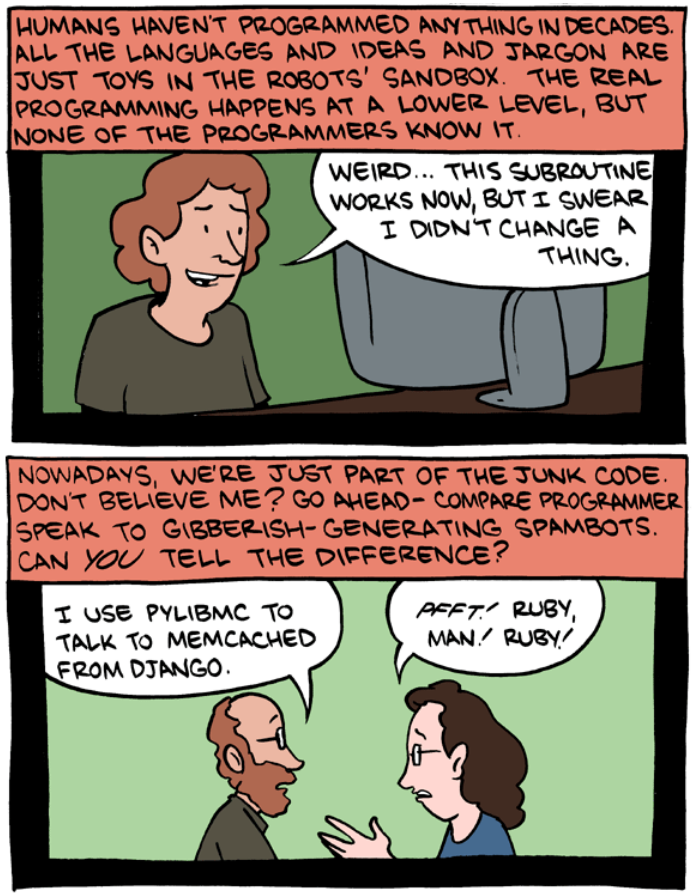
CSc 110, Spring 2017

Lecture 21: Reasoning about code

Adapted from slides by Marty Stepp and Stuart Reges



Punchline to a longer comic: http://www.smbc-comics.com/index.php?db=comics&id=2362#comic

Catch-up

Augmented assignment

Augmented assignment is the combination, in a single statement, of a binary operation and an assignment statement. -- docs.python.org

Augmented	Equivalent longer version
variable += value	**variable = variable + value**
variable -= value	**variable = variable - value**
variable *= value	**variable = variable * value**
variable /= value	**variable = variable / value**
variable %= value	**variable = variable % value**

```
x += 3                  # x = x + 3
gpa -= 0.5              # gpa = gpa - 0.5
number *= 2             # number = number * 2
```

Tally solution

Can we modify this to use augmented assignment?

```
# Returns the digit value that occurs most frequently in n.
# Breaks ties by choosing the smaller value.
def most_frequent_digit(n):
    counts = [0] * 10
    while (n > 0):
        digit = n % 10    # pluck off a digit and tally it
        counts[digit] = counts[digit] + 1
        n = n // 10

    ...
```

Tally solution with augmented assignment

```
# Returns the digit value that occurs most frequently in n.
# Breaks ties by choosing the smaller value.
def most_frequent_digit(n):
    counts = [0] * 10
    while (n > 0):
        digit = n % 10    # pluck off a digit and tally it
        counts[digit] += 1
        n //= 10

    ...
```

Reasoning about code

Reasoning about code

We can ask when a certain condition holds at a particular point in code.

Consider this code:

```
if (x >= 3):
    # --- Point A ---
    x -= 1
else:
    # --- Point B ---
    x += 1
    # --- Point C ---
# --- Point D ---
```

What do we know about `x`'s value at each of the four points?

When is `x > 3`? Always? Sometimes? Never?

More reasoning about code

- Consider the following condition at each point: when is `number < 0`?
  (Always? Sometimes? Never?)

```
number = int(input("Type a nonnegative number: "))
# Point A: is number < 0 here?

while (number < 0):
    # Point B: is number < 0 here?
    number = int(input("Negative; try again: "))

    # Point C: is number < 0 here?

# Point D: is number < 0 here?
```

Reasoning about code

- Right after a variable is initialized, its value is known:

```
x = 3
# is x > 0?
```

- In general we know nothing about parameters' values:

```
def mystery(a, b):
# is a == 10?
```

- But inside an `if`, `while`, etc., we may know something:

```
def mystery(a, b):
    if (a < 0):
        # is a == 10?
        ...
```

Reasoning about loops

- At the start of a loop's body, the loop's test must be `True`:

```
while (y < 10):
    # is y < 10?
    ...
```

- After a loop, the loop's test must be `False`:

```
while (y < 10):
    ...

# is y < 10?
```

- Inside a loop's body, the loop's test may become `False`:

```
while (y < 10):
    y += 1
    # is y < 10?
```

"Sometimes"

- Things that cause a variable's value to be unknown (often leads to “Sometimes" answers):
  - reading a value with `input()`
  - generating a number with `random()` or `randint()`
  - parameter initialization due to a function call

- If you can reach a point in the program with the answer sometimes being "yes" and sometimes being "no", then the correct answer is "sometimes."

Practice example 1

```
def mystery(x, y):
    z = 0
    # Point A

    while (x >= y):
        # Point B
        x = x - y
        z += 1

        if (x != y):
            # Point C
            z = z * 2
        # Point D
    # Point E
    print(z)
```

When are the following conditions true at the indicated points in the code? Choose ALWAYS, NEVER, or SOMETIMES.

	`x < y`	`x == y`	`z == 0`
Point A			
Point B			
Point C			
Point D			
Point E			

Practice example 2

```
def mystery():
    prev = 0
    count = 0
    next = int(input())
    # Point A

    while (next != 0):
        # Point B

        if (next == prev):
            # Point C
            count += 1

        prev = next
        next = int(input())
        # Point D
    # Point E
    return count
```

When are the following conditions true at the indicated points in the code? Choose ALWAYS, NEVER, or SOMETIMES.

	`next == 0`	`prev == 0`	`next == prev`
Point A			
Point B			
Point C			
Point D			
Point E			

Practice example 3

```
# Assumes y >= 0, and returns x^y
def pow(x, y):
    prod = 1

    # Point A
    while (y > 0):
        # Point B
        if (y % 2 == 0):
            # Point C
            x = x * x
            y = y // 2
            # Point D
        else:
            # Point E
            prod = prod * x
            y -= 1
            # Point F
    # Point G
    return prod
```

When are the following conditions true at the indicated points in the code? Choose ALWAYS, NEVER, or SOMETIMES.

	`y > 0`	`y % 2 == 0`
Point A	SOMETIMES	SOMETIMES
Point B	ALWAYS	SOMETIMES
Point C	ALWAYS	ALWAYS
Point D	ALWAYS	SOMETIMES
Point E	ALWAYS	NEVER
Point F	SOMETIMES	ALWAYS
Point G	NEVER	ALWAYS

Lists, indexes, and mappings

Count Vowels

- Write a function `vowel_count(s)` that accepts a string `s` as a parameter and returns a list of integers representing the counts of each vowel of string `s`.

- There are five vowels: a, e, i, o, u

- What data mapping would help to count the vowels?

- Write a helper function that returns a number representing the index of a vowel in the above mapping.

Vowel helper function

```
# Maps the characters a,e,i,o,u to the numbers 0,1,2,3,4,
# respectively. If the parameter c is not a vowel, returns -1
 def is_vowel(c):
    if (c == "a"):
        return 0
    elif (c == "e"):
        return 1
    elif (c == "i"):
        return 2
    elif (c == "o"):
        return 3
    elif (c == "u"):
        return 4
    return -1                  # parameter c is not a vowel
```

Count vowels

```
def main():
    vlist = vowel_count("i think, therefore i am")
    print("vlist = ", vlist)

# Return a list containing the counts of the number of vowels
# in string s
def vowel_count(s):
    # indices of list vowels map to a, e, i, o, u
    vowels = [0] * 5
    s = s.lower()
    for c in s:
        i = is_vowel(c)
        if (i >= 0):
            vowels[i] += 1
    return vowels
```