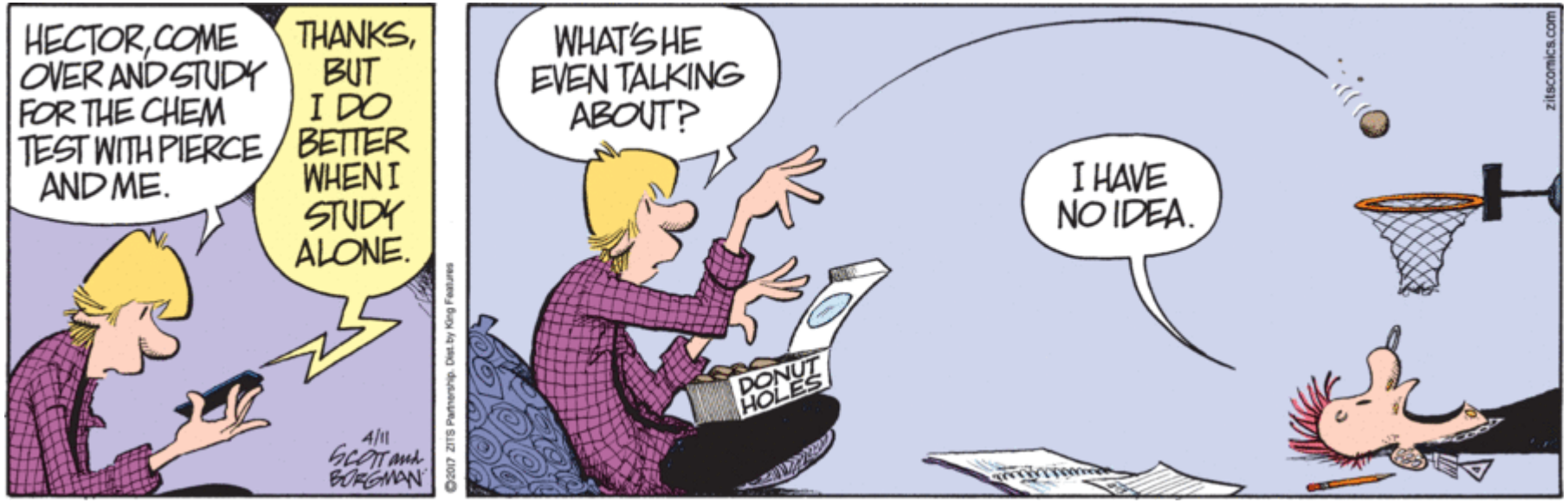
CSc 110, Spring 2018

Lecture 16: Fencepost Loops and `while` loops

Adapted from slides by Marty Stepp and Stuart Reges



A deceptive problem…

- Write a method `print_letters` that prints each letter from a word separated by commas.

  For example, the call:

  ```
  print_letters("Atmosphere")
  ```

  should print:

  ```
  A, t, m, o, s, p, h, e, r, e
  ```

Flawed solutions

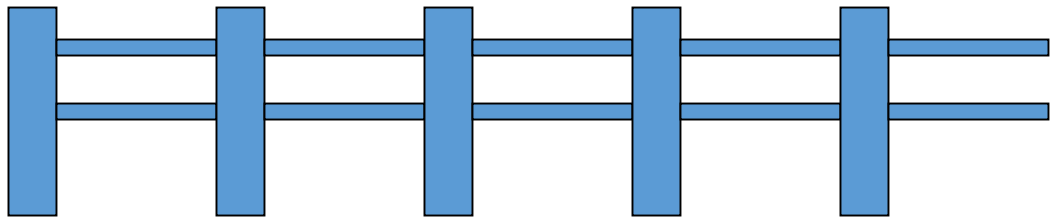
- ```
  def print_letters(word):
      for i in range(0, len(word)):
          print(word[i] + ", ", end='')
      print()    # end line
  ```
  - Output: A, t, m, o, s, p, h, e, r, e,

- ```
  def print_letters(word):
      for i in range(0, len(word)):
          print(", " + word[i], end='')
      print()    # end line
  ```
  - Output: , A, t, m, o, s, p, h, e, r, e

Fence post analogy

- We print *n* letters but need only *n* - 1 commas.
- Similar to building a fence with wires separated by posts:
  - If we use a flawed algorithm that repeatedly places a post + wire, the last post will have an extra dangling wire.

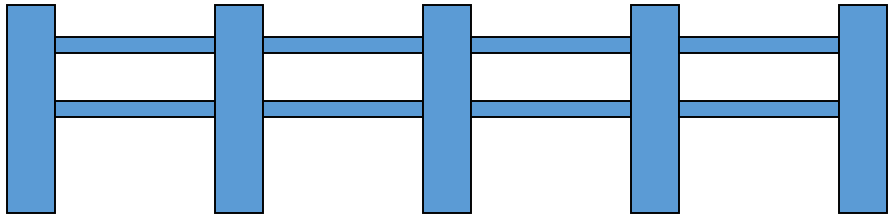
for length of fence :
    place a post.
    place some wire.



Fencepost loop

- Add a statement outside the loop to place the initial "post."
  - Also called a *fencepost loop* or a "loop-and-a-half" solution.

place a post.
for length of fence ***– 1****:*
    place some wire.
    place a post.



Fencepost function solution

- ```
  def print_letters(word):
      print(word[0])
      for i in range(1, len(word)):
          print(", " + word[i], end='')
      print()   # end line
  ```

- Alternate solution: Either first or last "post" can be taken out:

```
def print_letters(word):
    for i in range(0, len(word) - 1):
        print(word[i] + ", ", end='')
    last = len(word) - 1
    print(word[last]) # end line
```
```

Categories of loops

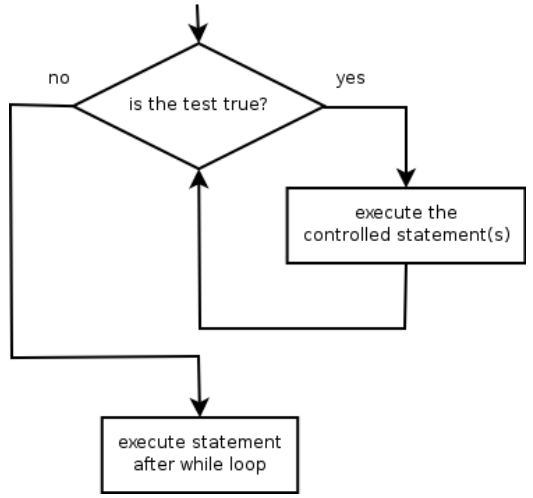
- **definite loop**: Executes a known number of times.
  - The `for` loops we have seen are definite loops.
    - Print "hello" 10 times.
    - Find all the prime numbers up to an integer *n*.
    - Print each odd number between 5 and 127.

- **indefinite loop**: One where the number of times its body repeats is not known in advance.
    - Prompt the user until they type a non-negative number.
    - Print random numbers until a prime number is printed.
    - Repeat until the user has typed "q" to quit.

The `while` loop

- **`while` loop**: Repeatedly executes its body as long as a logical test is true.

```
while test:
    statement(s)
```



- Example:

```
num = 1                                 # initialization
while num <= 200:                       # test
    print(str(num) + " ", end='')
    num = num * 2                       # update

# output:  1 2 4 8 16 32 64 128
```

Example `while` loop

```
# finds the first factor of 91, other than 1
n = 91
factor = 2
while n % factor != 0:
    factor += 1
print("First factor is", factor)

# output:  First factor is 7
```

- `while` is better than `for` because we don't know how many times we will need to increment to find the factor.

Sentinel values

- **sentinel**: A value that signals the end of user input.
  - **sentinel loop**: Repeats until a sentinel value is seen.

- Example: Write a program that prompts the user for text until the user types "quit", then output the total number of characters typed.
  - (In this case, "quit" is the sentinel value.)

```
Type a word (or "quit" to exit): hello
Type a word (or "quit" to exit): yay
Type a word (or "quit" to exit): quit
You typed a total of 8 characters.
```

Solution?

```
sum = 0
response = "dummy"    # "dummy" value, anything but "quit"

while response != "quit":
    response = input("Type a word (or \"quit\" to exit): ")
    sum += len(response)

print("You typed a total of " + str(sum) + " characters.")
```

- This solution produces the wrong output. Why?

```
You typed a total of 12 characters.
```

The problem with our code

- Our code uses a pattern like this:

  sum = 0
  while input is not the sentinel:
  prompt for input; read input.
  add input length to the sum.

- On the last pass, the sentinel's length (4) is added to the sum:

  prompt for input; read input (`"quit"`*).*
  add input length (4) to the sum.

- This is a fencepost problem.
  - Must read *N* lines, but only sum the lengths of the first *N*-1.

A fencepost solution

```
sum = 0.
prompt for input; read input.              # place a "post"

while (input is not the sentinel):
    add input length to the sum.           # place a "wire"
    prompt for input; read input.          # place a "post"
```

- Sentinel loops often utilize a fencepost "loop-and-a-half" style solution by pulling some code out of the loop.

Correct code

```
sum = 0

# pull one prompt/read ("post") out of the loop
response = input("Type a word (or \"quit\" to exit): ")

while (response != "quit"):
    sum += len(response)    # moved to top of loop
    response = input("Type a word (or \"quit\" to exit): ")

print("You typed a total of " + str(sum) + " characters.")
```

Sentinel as a constant

```
SENTINEL = "quit"
...

sum = 0

# pull one prompt/read ("post") out of the loop
response = input("Type a word (or \"" + SENTINEL + "\" to exit): ")

while response != SENTINEL:
    sum += len(response)    # moved to top of loop
    response = input("Type a word (or \"" + SENTINEL + "\" to exit): ")

print("You typed a total of " + str(sum) + " characters.")
```