Designing Loops: Basis and Progress

Reasoning about loops

Some loops are easy:
- Do ... to each of these

Some loops are not so easy:
- Keeping doing ... until ... um, uh, until ...
- something!

How can we think about the more difficult loops, to devise them and get them right?

While loops: Two parts

Basis (base case)
- One or more small, simple cases, which can be done in one step without looping.

Progress
- A case that requires a simple step, and then solution of a smaller problem.

Inductive while loop structure

while work to be done:
  if simplest case:
    just do it
  else:
    do one step
    prepare for next step

Base case
Progress
**Inductive while loop structure**

While ( work to be done ) {
    if (simplest case) {
        just do it;
    } else {
        do one step;
        prepare for next step;
    }
}

**Example: Palindrome checker**

Palindrome: Same forward and backward example: rotator

```python
def isPalindrome(s):
    #
    We know we'll need to loop through the letters in s, but how?
```

**String methods we can use**

len(s) tells us how many characters in s

s[i] lets us look at the ith character of s
s[0] is the first character in s
s[-1] is the last character
s[m:n] is a slice; s[1:-1] chops off first and last
String methods we can use

s.length() tells us how many characters in s

s.charAt(i) let’s us look at the i\text{th} character of s

s.charAt(0) is the first character in s
s.charAt( s.length() – 1 ) is the last character

Palindrome loop design

What’s the base case
  (in terms of fingers on the letters)

What’s the progress case
  (in terms of fingers on the letters)
Progress: So far so good

```
LEVEL
```

Palindrome pseudocode, first try

Place fingers on first and last letter
while (forever) {
    if fingers are on the same letter { return true }
    if fingers have crossed { return true }
    if letter under left finger ≠ letter under right 
        { return false }
    else { move fingers toward the center }
}

A little better ...

Place fingers on first and last letter
while (left finger < right finger) {
    if letter under left finger ≠ letter under right 
        { return false }
    else { move fingers toward the center }
}
return true

And just a little simpler ...

Place fingers on first and last letter
while (left finger < right finger) {
    if letter under left finger ≠ letter under right 
        { return false }
    move fingers toward the center
}
return true
Into Python ... (version 1)

```python
// Place fingers on first and last letter
left = 0
right = len(s) - 1
while (left finger < right finger) {
    if letter under left finger ≠ letter under right
        return false;
    move fingers toward the center
}
return true;
```

Into Java ...

```java
// Place fingers on first and last letter
int left = 0;
int right = s.length() - 1;
while (left < right) {
    if letter under left finger ≠ letter under right
        return false;
    move fingers toward the center
}
return true;
```

Into Java ...

```java
// Place fingers on first and last letter
int left = 0;
int right = s.length() - 1;
while (left < right) {
    if letter under left finger ≠ letter under right
        return false;
    move fingers toward the center
}
return true;
```

Into Python ...

```python
# Place fingers on first and last letter
left = 0
right = len(s) - 1
while left < right:
    if s[left] != s[right]:
        return False
    move fingers toward the center
return True
```
### Into Python...

```python
# Place fingers on first and last letter
left = 0
right = len(s) - 1;
while left < right:
    if s[left] != s[right]:
        return False
    # Move fingers toward the center
    left = left + 1
    right = right - 1
return True
```

### Into Python... with slices

```python
# At each step, s is the portion between fingers
while len(s) > 1:
    if s[0] != s[-1]:
        return False
    # Move fingers toward the center
    s = s[1:-1]
return True
```

### Into java ...

```java
// Place fingers on first and last letter
int left = 0;
int right = s.length() - 1;
while (left < right) {
    if (s.charAt(left) != s.charAt(right)) {
        return false;
    }
    // Move fingers toward the center
    left = left + 1;
    right = right - 1;
}
return true;
```

### Into java ...

```java
// Place fingers on first and last letter
int left = 0;
int right = s.length() - 1;
while (left < right) {
    if (s.charAt(left) != s.charAt(right)) {
        return false;
    }
    left = left + 1; // left finger moves right
    right = right - 1; // right finger moves left
}
return true;
```
As Java ...

```java
/**
 * Determine whether s is a palindrome.
 * @param s string to be checked
 * @return true if and only if s is a palindrome
 */
static boolean isPalindrome(String s) {
    // Place fingers on first and last letter
    int left = 0;
    int right = s.length() - 1;
    while (left < right) {
        if (s.charAt(left) != s.charAt(right)) {
            return false;
        }
        left = left + 1; // left finger moves right
        right = right - 1; // right finger moves left
    }
    return true;
}
```

But is it correct?

Argument for correctness follows case breakdown

**Basis:** Should handle all problems up to some size, e.g., “words of up to 1 character”

**Progress:** Should handle all other problems, and the “prepare for next step” part should produce a smaller problem (e.g., a shorter substring between left and right finger)

You’ve probably seen this before

(Though you may not have liked it)

Basis, progress cases are exactly
Base case, inductive case in proof by induction

Does this make you happy or sad?

Either way, it’s the truth ... we reason about loops with inductive proofs.

Summary

**Easy loops**
Do something to each ...

*Example:* Do something to each word in a text file

**Harder loops**
Break down into cases: Basis, Progress

*Design conceptual logic first, then code*
Inductive *while* loop structure

While (work to be done) {
    if (simplest case) {
        just do it;
    } else {
        do one step;
        prepare for next step;
    }
}

*Java syntax or Python syntax ... the fundamental logic is the same.*