Announcements

- A4: Due 4/20 at 11:59pm
- Thursday 4/20: Review session in lecture
- Prelim 2 on Tuesday 4/25, 7:30pm – 9pm
  * Covers material up through Tuesday 4/18
  * Lecture: Professor office hours
  * Labs: TA/consultant office hours
- No labs on 4/26

The isinstance Function

- isinstance(<obj>,<class>)
  * True if <obj>’s class is same as or a subclass of <class>
  * False otherwise

  - Example:
    - isinstance(e,Executive) is True
    - isinstance(e,Employee) is True
    - isinstance(e,object) is True
    - isinstance(e,str) is False
  
  - Generally preferable to type
    - Works with base types too!

Beyond Sequences: The while-loop

**while <condition>:**

- Relationship to for-loop
  * Broader notion of “still stuff to do”
  * Must explicitly ensure condition becomes false
  * You explicitly manage what changes per iteration

- Definition:
  - Since you manage changes yourself
  - Can state changes per iteration

- Relationship to for: broader “still stuff to do” notion
  * Must explicitly ensure condition becomes false
  * You manage changes per iteration

- While-Loops and Flow

```python
print 'Before while'
count = 0
i = 0
while i < 3:
    print 'Start loop '+str(i)
    count = count + i
    i = i + 1
print 'End loop '+str(i)
print 'After while'
```

Output:

```
Before while
Start loop 0
Start loop 1
Start loop 2
After while
```

while Versus for

```python
# process range b..c-1
for k in range(b,c):
    # code involving k
```

```python
# process range b..c-1
for k in range(b,c+1):
    # code involving k
```

Note on Ranges

- m..n is a range containing n+1-m values
  * 2..5 contains 2, 3, 4, 5.
  * Contains 5+1 – 2 = 4 values
  * 2..4 contains 2, 3, 4.
  * Contains 4+1 – 2 = 3 values
  * 2..3 contains 2, 3.
  * Contains 3+1 – 2 = 2 values
  * 2..2 contains 2.
  * Contains 2+1 – 2 = 1 values
  * 2..1 contains ???

- Notation m..n always implies that m <= n+1
  * If m = n+1, the range has 0 values
Patterns for Processing Integers

\[ \text{range } a..b-1 \]

\[ i = a \]
\[ \text{while } i < b: \]
\[ \quad \# \text{process integer } i \]
\[ \quad i = i + 1 \]
\[ \# \text{store in count # of } '/' \text{s in } \text{String } s \]
\[ \text{count} = 0 \]
\[ i = 0 \]
\[ \text{while } i < \text{len}(s): \]
\[ \quad \text{if } s[i] == '/': \]
\[ \quad \quad \text{count} = \text{count} + 1 \]
\[ \quad i = i + 1 \]
\[ \# \text{count is # of } '/' \text{s in } s[0..s.length()-1] \]

\[ \text{range } c..d \]

\[ i = c \]
\[ \text{while } i <= d: \]
\[ \quad \# \text{process integer } i \]
\[ \quad i = i + 1 \]
\[ \# \text{Store in double var. } v \text{ the sum } \]
\[ \quad \text{1/1 + 1/2 + ... + 1/n} \]
\[ v = 0; \quad \# \text{call this 1/0 for today} \]
\[ i = 0 \]
\[ \text{while } i <= n: \]
\[ \quad v = v + 1.0 / i \]
\[ \quad i = i + 1 \]
\[ \# v = 1/1 + 1/2 + ... + 1/n \]

while Versus for

\[ \# \text{table of squares to } N \]
\[ \text{seq} = [] \]
\[ n = \text{floor}(\text{sqrt}(N)) + 1 \]
\[ \text{for } k \text{ in range}(n): \]
\[ \quad \text{seq}.append(k*k) \]

\[ \# \text{table of squares to } N \]
\[ \text{seq} = [] \]
\[ k = 0 \]
\[ \text{while } k*k <= N: \]
\[ \quad \text{seq}.append(k*k) \]
\[ k = k+1 \]

A for-loop requires that you know where to stop the loop \textbf{ahead of time}.

A while loop can use complex expressions to check if the loop is done.

Cases to Use while

Great for when you must \textbf{modify} the loop variable

\[ \# \text{Remove all 3's from list } t \]
\[ i = 0 \]
\[ \text{while } i < \text{len}(t): \]
\[ \quad \# \text{no 3's in } t[0..i-1] \]
\[ \quad \text{if } t[i] == 3: \]
\[ \quad \quad \text{del } t[i] \]
\[ \quad \text{else:} \]
\[ \quad \quad i += 1 \]

The stopping condition is not a numerical counter this time.

Simplifies code a lot.

Some Important Terminology

\- \textbf{assertion}: true-false statement placed in a program to \texttt{assert} that it is true at that point
\- Can either be a comment, or an \texttt{assert} command
\- \textbf{invariant}: assertion supposed to "always" be true
\- If temporarily invalidated, must make it true again
\- \textbf{Example}: class invariants and class methods
\- \textbf{loop invariant}: assertion supposed to be true before and after each iteration of the loop
\- \textbf{iteration of a loop}: one execution of its body

Preconditions & Postconditions

\[ \# x = \text{sum of } 1..n-1 \]
\[ x = x + n \]
\[ n = n + 1 \]
\[ \# x = \text{sum of } 1..n-1 \]

\textbf{Precondition}: assertion placed before a segment

\textbf{Postcondition}: assertion placed after a segment

Relationship Between Two

If precondition is true, then postcondition should be true