More about loops
Homework 3

• Three musketeers game

• Mostly practice for your list skills

• Also first homework with unit tests
  – We have given you all the unit tests you need to implement
while loops

- The while loop is the simplest kind of loop
  - The **test** is performed before the statements are executed
  - Thus, if the **test** is initially **False**, the while loop does nothing
  - If the test is **True**, the statements are executed, then the test is performed again

- The statements must eventually cause the test to become **False**, else you have a so-called **infinite loop**
while loop initialization

• A common idiom is to set something up before the while loop, and tweak it at the bottom of the loop

  get some value
  while something_about_the_value:
    do some things with the value
  get another value

• Example:

  password = input(“Enter your password: “)
  while password != actual_password:
    print(“That’s not your password!”)
  password = input(“Enter your password: “)
for loops

• **for** loops execute their statements for a fixed number of values, setting the **loop index** to each value in turn

• The values can be in the form of a **list**

```python
names = ["Tom", "Dick", "Harry"]
for name in names:
    print(name)
```

• The values can be in the form of a **set**

```python
names = {"Tom", "Dick", "Harry"}
for name in names:
    print(name)
```

• The values can be in the form of a **dictionary**

```python
names = ["Tom": 25, "Dick": 23, "Harry": 25]
for name in names:  # steps through the keys
    print(name, "->", names[name])
```
for loops, cont’d

• **for** loops execute their statements for a fixed number of values, setting the **loop index** to each value in turn
• The values can be given by an **iterator**, which is a function that provides values as needed
• The most common iterator is **range**
  – **range(start, end)** produces integer values starting with **start** and going up to, but not including, **end**
  – **range(end)** is equivalent to **range(0, end)**
  – **range(start, end, step)** produces integer values starting with **start** and going up by steps of **step** up to, but not equaling or exceeding, **end**
  – IDLE example: range.py
• The **break** statement is used to exit a loop early
• IDLE example: break.py
• Many programmers feel it is bad style to ever use a break
  – I recommend using a break only as a last resort, if you can’t figure out a better way to exit a loop normally
continue

• The `continue` statement is used to skip the rest of the loop and go back to the top
• IDLE example: continue.py
• Like `break`, `continue` really only makes sense within an if statement
  – While not as bad as `break`, many programmers don’t like to use `continue`
  – Think about alternatives before using a `continue`
pass

- The **pass** statement is the easiest of all – it does nothing
- **pass** is used mostly as a placeholder, where a statement is required but you haven’t yet figured out what to do there
- IDLE example: pass.py