

Numeric For statement

Alternative form of the for statement.

Most commonly used to step through a sequence of numbers

Can be used more generally than this.

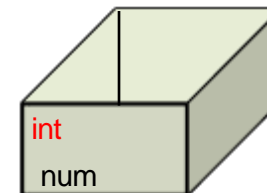
More tricky to get right than the for each version.

Four components

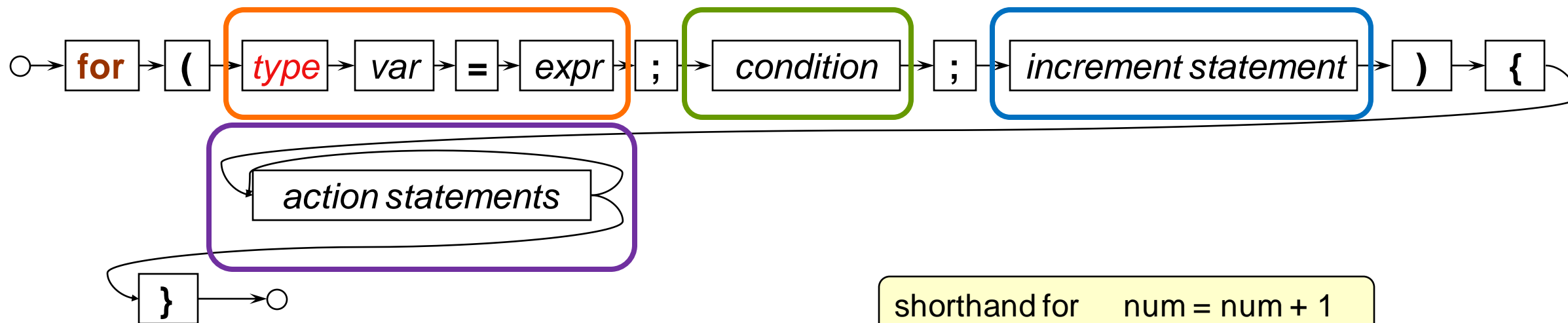
- a variable and its initial value.
- a condition when to keep going / stop
- how to increment the variable each time
- actions to perform for each time

```
// print each number from 1 to 100:
```

```
for ( int num =1 ; num <= 100 ; num = num + 1 ) {  
    UI.println(num);  
}
```



For statement ("Numeric" version)



```

for ( int num = 0 ; num < 1000 ; num++ ) {
    UI.println(num);
}
  
```

• Meaning:

- initialise the variable
- repeat, as long as the condition is true:
 - do the actions
 - do the increment

Using Numeric For: #1

- Print a table of numbers and their squares:

```
public void printTable(int max){  
    UI.println("Table of integers and their squares");  
    for (int num = 1; num <= max; num = num + 1 ) {  
        UI.printf(" %3d  %6d \n", num, (num*num));  
    }  
}
```

Using Numeric For: #2

Doesn't have to increment by 1 each time:

```
/**
 * Print each even number between start and end (inclusive)
 */
public void printEvenNumbers(int start, int end ){
    if (start%2==1 ) { // make sure start is even
        start = start + 1;
    }
    for ( int num = start; num <= end; num = num + 2 ) {
        UI.println(num);
    }
}
```

Using Numeric For: #3

- Draw a row of squares:

```
public static final double SIZE = 20;
```

```
:
```

```
/** Draws count squares in a horizontal row, starting at (left,top) */
```

```
public void drawRowOfSquares (double left, double top, int count){
```

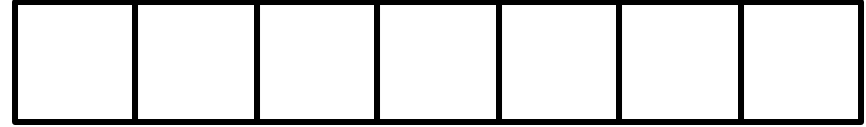
```
    for (int i = 0; i < count; i++ ) {
```

```
        double x = left + i * SIZE;
```

```
        UI.drawRect(x, top, SIZE, SIZE);
```

```
    }
```

```
}
```

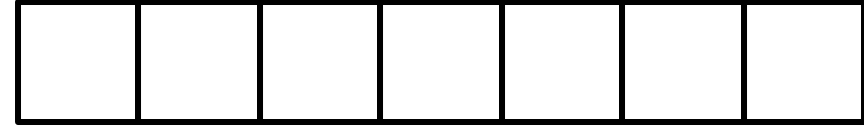


i++
is shorthand for
i = i + 1

Counting from 0 is often easier,
especially for drawing stuff!

Using Numeric For: #4

- Draw a row of squares:



```
public static final double SIZE = 20;
```

```
⋮
```

```
/** Draws count squares in a horizontal row, starting at (left,top) */
```

```
public void drawRowOfSquares (double left, double top, int count){
```

```
    double right = left+count*SIZE;
```

```
    for (double x = left; x < right; x = x + SIZE) {
```

```
        UI.drawRect(x, top, SIZE, SIZE);
```

```
    }
```

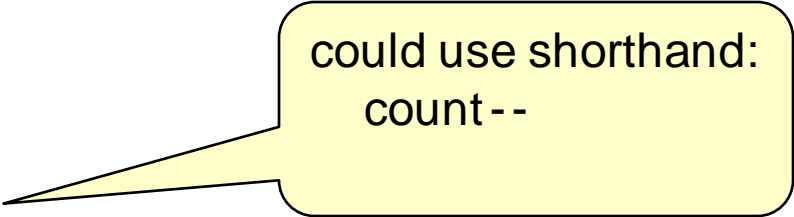
```
}
```

Note: this for statement is stepping through a sequence of doubles, rather than ints.

Using Numeric For: #5

- For doesn't have to step up:

```
public void countDown(int start){
    UI.setFontSize(100);
    for (int count = start; count >= 1; count = count - 1) {
        UI.clearGraphics();
        UI.drawString( ""+count, 200, 300 );
        UI.sleep(500);
    }
    UI.clearGraphics();
    UI.setColor(Color.red);
    UI.drawString("GO", 200, 300);
}
```



could use shorthand:
count--

Count from 0 or 1?

Counted for loop: Can count from 0 or from 1

```
for (int n = 0; n < target; n++) {  
    <do actions>  
}  
OR  
for (int n = 1; n <=max; n++) {  
    <do actions>  
}
```

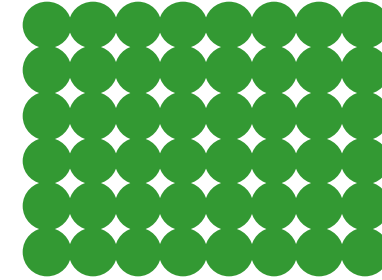
- If counting from 0,
 - n is the number of iterations that have been completed
 - Loop as long as n is **less than** target:
 - Good for drawing
 - Good for dealing with lists and arrays.
- If counting from 1,
 - n is the iteration it is about to do
 - Loop as long as n is **less than or equal to** target:

Off-by-one errors are common when you mix these two up.

Nested for loops

Can have loops inside loops:

eg: Draw a grid of circles



```
public void drawCircles(int rows, int cols, int diam) {
```

```
    for (int row = 0; row < rows; row++) {
```

```
        double y = TOP + row*diam;
```

```
            for (int col = 0; col < cols; col++) {
```

```
                double x = LEFT + col*diam;
```

```
                UI.fillOval(x, y, diam, diam);
```

```
            }
```

```
    }
```

```
}
```

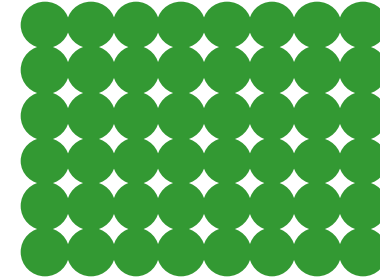
Outside loop:
do each row

Inside loop:
do each column within the
current row

Nested for loops

Nested loops can be row first, or column first:

eg Draw a grid of circles (by column)



```
public void drawCircles(int rows, int cols, int diam) {
```

```
    for (int col = 0; col < cols; col++) {
```

```
        double x = LEFT + col*diam;
```

```
        for (int row = 0; row < rows; row++) {
```

```
            double y = TOP + row*diam;
```

```
            UI.fillOval(x, y, diam, diam);
```

```
        }
```

```
    }
```

```
}
```

Outside loop:
do each column

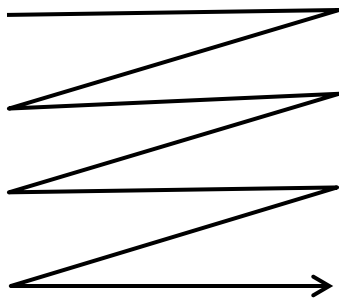
Inside loop:
do each row within the
current column

Designing nested loops with numbers

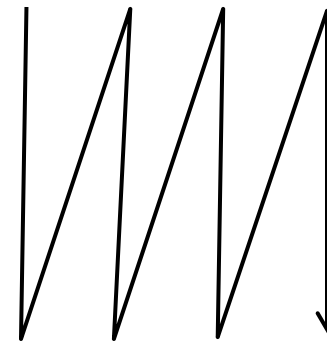
2D structures, eg table of rows and columns:

- Can do rows in the outside loop and columns in the inside loop, or vice versa

```
for (int row=0; row<rows; row++) {
    for (int col=0; col<cols; col++) {
        <do actions for row, col >
    }
}
```



```
for (int col=0; col<cols; col++) {
    for (int row=0; row<rows; row++) {
        <do actions for row, col >
    }
}
```



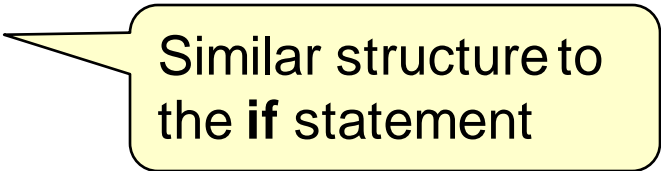
Today

- Repetition/Iteration
 - repeating something as long as a condition stays true ("while" loop)
- Test:
 - 28 October - Lecture Time (Mingli 5-301)

While statements: repeating with a condition

- **For** statements: repetition over a list of values or a sequence of numbers.
- **While** statements : general repetition, subject to a condition.

```
while (condition-to-do-it-again) {  
    actions to perform each time round  
}
```

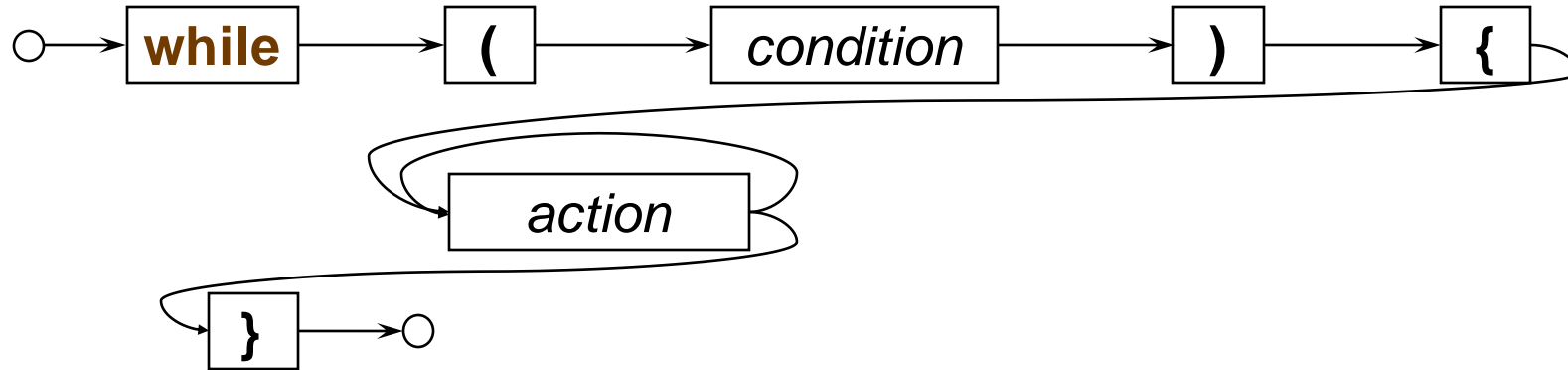


Similar structure to
the **if** statement

```
while ( true ) {  
    UI.println("this repeats forever!");  
}
```

- Similar to **for**, but NOT THE SAME!
 - same condition and actions;
 - no built-in initialisation and increment.
 - Appropriate if you don't know how many times it will repeat

While statement



- Meaning:
 - Repeatedly
 - If the condition is still true, do the actions another time
 - If the condition is false, stop and go on to the next statement.
 - Note: don't do actions at all if the condition is initially false
- Similar to **if**, but NOT THE SAME!
 - keeps repeating the actions,
 - as long as the condition is still true each time round
 - no **else** — just skips to next statement when condition is false

While is a rearrangement of for

- Print a table of numbers and their squares:

```

public void printTable(int max){
    int num = 1;           Initialise
    while ( num <= max ) { Test
        Ul.printf(" %3d  %6d  \n", num, (num*num)); Body
        num = num + 1;     Increment
    }
}

```

- Repetition with **while** generally involves

- initialisation: get ready for the loop Put before while loop
- test: whether to repeat
- body: what to repeat
- “increment”: get ready for the next iteration Put at end of actions.

General while loops

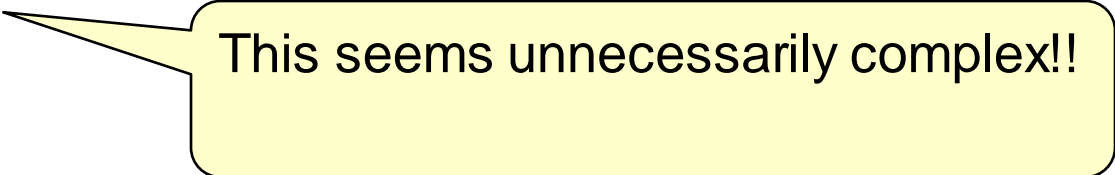
```
/** Practice times-tables until got 5 answers correct in a row */
```

```
public void playArithmeticGame () {
    int score = 0;
    while ( score < 5 ) {
        // ask an arithmetic question
        int a = this.randomInteger(10);
        int b = this.randomInteger(10);
        int ans = UI.askInteger("What is " + a + " times " + b + "?");
        if ( ans == a * b ) {
            score = score + 1;
        }
        else {
            score = 0;
        }
    }
    UI.println("You got 5 right answers in a row");
}
```

```
/** return random int between 1 and max (inclusive) */
public int randomInteger(int max) {
    return (int) (Math.random() * max ) + 1;
}
```


General while loops

```
/** Ask a multiplication problem until got it right */
public void practiceArithmetic (){
    int a = this.randomInteger(10);
    int b = this.randomInteger(10);
    String question = "What is " + a + " times " + b + "?";
    boolean correct = false;
    while ( ! correct) {
        int ans = UI.askInteger(question);
        if ( ans == a * b ) {
            correct = true;
            UI.println("You got it right!");
        }
        else {
            UI.println("sorry, try again");
        }
    }
}
```



This seems unnecessarily complex!!

Loops with the test "in the middle"

If the condition for exiting the loop depends on the actions, need to exit in the middle!

Common with loops asking for user input.

- **break** allows you to exit a loop (**while**, or **for**)
 - Must be inside a loop
 - Ignores any **if** 's
 - Does not exit the method (**return** does that)

```
while ( true ) {  
    actions to set up for the test  
    if ( exit-test ) {  
        break;  
    }  
    additional actions  
}
```

General while loops with break

```
/** Ask a multiplication problem until got it right */
```

```
public void practiceArithmetic () {
```

```
    int a = this.randomInteger(10);
```

```
    int b = this.randomInteger(10);
```

```
    String question = "What is " + a + " times " + b + "?";
```

```
    while ( true ) {
```

```
        int ans = UI.askInteger(question);
```

Setting up for test

```
        if ( ans == a * b ) {
```

```
            UI.println("You got it right!");
```

Test and break

```
            break;
```

```
        }
```

```
        UI.println("sorry, try again");
```

actions if test failed.

```
    }
```

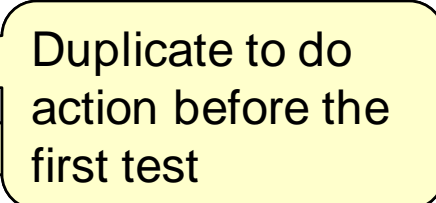
```
}
```

- Only use **break** when the exit is not at the beginning of the loop.

While loops to get valid input

```
/** Ask for an email address and insist that it contains one @ in the middle */
```

```
public String askEmailAddress (){  
    String addr = UI.askString("Enter email address");  
    while ( ! this.validEmailAddress(addr) ){  
        UI.println("Email address must have a single @ with characters before and afterwards");  
        addr = UI.askString("Enter email address");  
    }  
    return addr;  
}
```



Duplicate to do action before the first test

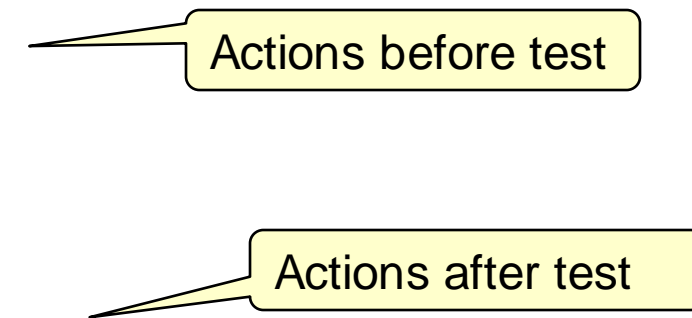
```
/** Check that an email address contains one @ in the middle and no spaces */
```

```
public boolean validEmailAddress (String addr){  
    int idx = addr.indexOf("@");  
    return (idx>0 && idx!=addr.length()-1 && addr.indexOf("@", idx+1)==-1 && !addr.contains(" "));  
}
```

While loops to get valid input

```
/** Ask for an email address and insist that it contains one @ in the middle */
```

```
public String askEmailAddress (){  
    while (true) {  
        String addr = UI.askString("Enter email address");  
        if ( this.validEmailAddress(addr) ) {  
            break;  
        }  
        UI.println("Email address must have a single @ with characters before and afterwards");  
    }  
    return addr;  
}
```



While loops to get valid input

```
/** Ask for an email address and insist that it contains one @ in the middle */
```

```
public String askEmailAddress (){
```

```
    while (true) {
```

```
        String addr = UI.askString("Enter email address");
```

Actions before test

```
        if ( this.validEmailAddress(addr) ) {
```

```
            return addr;
```

```
        }
```

Actions after test

```
        UI.println("Email address must have a single @ with characters before and afterwards");
```

```
    }
```

```
}
```

- Don't need **break** if there is nothing to do after the loop except **return**:
 - just **return** from the middle of the loop!

More loops with user input

- Make user guess a magic word:

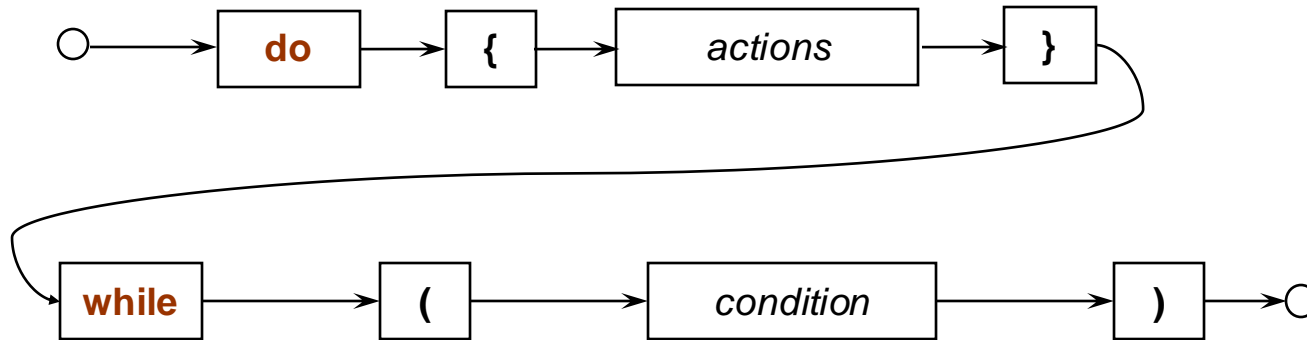
```
public void playGuessingGame(String magicWord){
    UI.println("Guess the magic word:");
    while (true) {
        String guess = UI.askString("your guess: ");
        if ( guess.equalsIgnoreCase(magicWord) ) {
            UI.println("You guessed it!");
            break;
        }
        UI.println("No, that wasn't right. Try again!");
    }
}
```

Setting up for test

Test and break

Additional actions

While statement



- Meaning:
 - Repeatedly do the actions while the condition is true
 - If the condition is false, stop and go on to the next statement.
- Similar to the “regular” **while**, but NOT THE SAME!
 - do actions once if the condition is initially false

do-while and while loops

- A do-while loop can be translated into a while loop:

```
int i = 0;
do {
    UI.println("I can count to " + i + "!");
    i++;
} while (i < 10)
```

```
String sentence = "";
do {
    String word = UI.askString("Next word: ");
    sentence = sentence + word;
} while (! sentence.endsWith("."))
```

```
initialisation
do {
    body
    change
} while (condition)
```

```
int i = 0;
while (i < 10) {
    UI.println("I can count to " + i + "!");
    i++;
}
```

```
String sentence = UI.askString("Next word: ");
while (! sentence.endsWith(".")) {
    String word = UI.askString("Next word: ");
    sentence = sentence + word;
}
```

```
initialisation
while (condition) {
    body
    change
}
```

Designing loops

Is the number of steps determined at the beginning?

```
int i = 0;           int i = 1;
while ( i < number) { while ( i <= number) {
    <do actions>      <do actions>
    i = i + 1;        i++;
}                    }
```

- Note, can count from 0 or from 1!

Otherwise

```
<initialise>
while ( <condition-to-do it again> ) {
    <body>
    <increment>
}
```

Designing loops

- Write out the steps for a couple of iterations
 - including the tests to determine when to quit/keep going
- Identify the section that is repeated
 - preferably starting with the test
- Wrap it with a **while** () { }
- Identify the condition for repeating (and initial state).

Designing loops

- Write out the steps for a couple of iterations
 - including the tests to determine when to quit/keep going
- Identify the section that is repeated
 - preferably starting with the test
- Wrap it with a **while** () { }
- Identify the condition for repeating (and initial state).

More loops with user input – magic word

- Make user guess a magic word:
 - prompt user for a word
 - test if it is “pumpkin” => stop
 - if not
 - say no!
 - prompt user for a word
 - test if it is “pumpkin” => stop
 - if not
 - say no!
 - prompt user for a word
 - test if it is “pumpkin” => stop
 - if not
 - say no!
 - prompt user for a word
 - test if it is “pumpkin” => stop

More loops with user input – magic word

- Make user guess a magic word:
 - prompt user for a word
 - test if it is “pumpkin”
 - if not, try again

```
UI.print("Please enter the magic word:");
```

```
String answer = UI.askString("your guess: ");
```

```
if ( answer.equalsIgnoreCase("pumpkin") ) { ...
```

```
    if not, go back           where to??
```

```
at end
```

```
UI.println("You finally guessed it!");
```

Magic word 1: break to exit

- Put “while” at the beginning of the repeated section
- Use the “infinite loop” and an `if () { break; }`

```
while ( true ) {  
    UI.print(“Please enter the magic word:”);  
    String answer = UI.askString(“your guess: ”);  
    if ( answer.equalsIgnoreCase(“pumpkin”) ) {  
        break;  
    }  
    UI.print(“No!”);  
}  
UI.println(“You finally guessed it!”);
```

Magic word 2: unfold

- Put “while” where the test is,
- Repeat the “set up” in the body.

```
UI.print("Please enter the magic word:");  
String answer = UI.askString("your guess: ");  
while ( ! answer.equalsIgnoreCase("pumpkin") ) {  
    UI.println("No!");  
    answer = UI.askString("your guess: ");  
}  
UI.println("You finally guessed it!");
```


Magic word 3: clever initialisation

- Set up a “dummy” case first.

```
UI.print("Please enter the magic word:");  
String answer = "not pumpkin!";  
while ( ! answer.equalsIgnoreCase("pumpkin") ) {  
    answer = UI.askString("your guess: ");  
}  
UI.println("You finally guessed it!");
```

Testing your program

- A) Need to try out your program on sample input while removing the "easy" bugs.
 - Can be a pain if need lots of input (eg TemperatureAnalyser)
 - UI window has a menu item – "set input" – to get input from a text file instead of user typing it.
 - ⇒ don't have to type lots of data each time
 - Create the text file, eg in Notepad
 - Select file using menu *before* the program has started asking for input.
 - File can contain multiple sequences of data.

- B) Need to test your program on a range of inputs
 - Easy, "ordinary", inputs
 - Boundary cases — values that are only just in range, or just out of range
 - Need to check that your **if** conditions are right
 - Invalid data—does your program handle invalid input correctly?

Creating test cases involves creativity – have to try to come up with ways to break your program.