Programming 1 / Object Oriented Programming

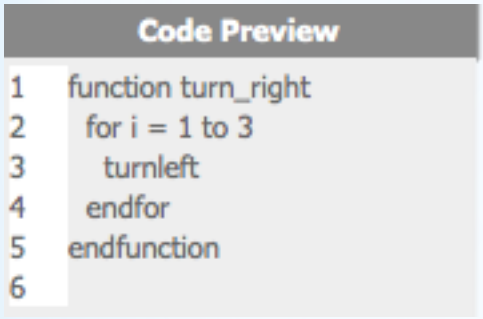
Thinking Like a Programmer: Lecture #4 -text-based programming in Banana

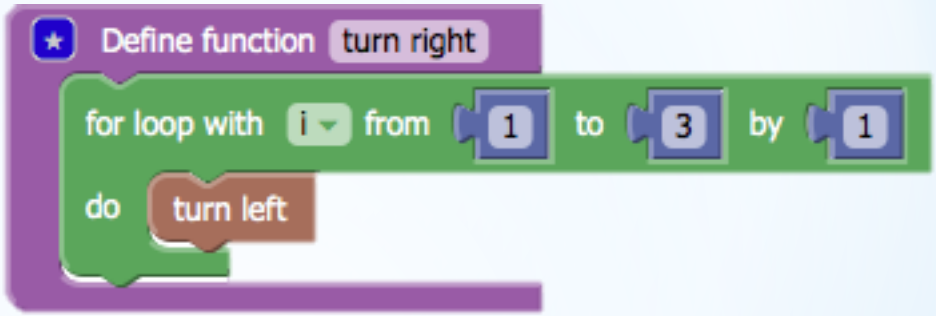
The story so far

- We have written programs to navigate Carol around a maze
  - These were visual problems
- We have written programs that interact with the user to get input and calculate results and produce output
  - These were abstract problems
- Until now we have been using our block based system of programming
- This has helped you get a head start and concentrate on problem solving – but now it's time to look at more conventional approaches to programming
- Today, and in this week's workshop, we'll be discarding the blocks and you'll be writing the text based code to solve your problems by hand

The story so far

- You've actually been creating text-based code in all of our activities so far…





- The block system generates text based code as you drag, drop and move blocks about the canvas
- The text based code is what NoobLab runs - not the blocks

Introducing Banana

- This text based code is a language of my own devising called *Banana*
- Why is it called Banana? Well, when I teach programming people often ask me what to call things in their code, e.g.
  - *Q: "What should I call my variable?"*
- To which I usually say
  - *A: "You can call it whatever you like… you can call it banana if you like!"*
- Banana is designed to be like *pseudocode*

Code Preview

```
1  function turn_right
2    for i = 1 to 3
3      turnleft
4    endfor
5  endfunction
6
```

What is pseudocode?

- Pseudocode is simpler than natural language, which helps make it less ambiguous
- It is half-way between natural language and very logically precise "real" computer languages
- There is no single "standard" of pseudocode – if you Google for examples you'll find all kinds of different things, e.g.
  - some use PRINT, some use DISPLAY
  - some use SET for variables, others use DEFINE
- This works because pseudocode is NOT a programming language – it's just a technique for helping you design and plan your programs before you write them in a real language
- So, pseudocode is not something you would type into a computer and run – you would just use it to map things out in advance

Banana and pseudocode

- Banana is *like* pseudocode – it uses many of the same principles and keywords
- If you can read and write Banana, you'll be able to read and write pseudocode
- We distinguish between Banana and pseudocode because
  - Banana code can be run on a computer – pseudocode is not
  - There are rules and standards for Banana code – there are none for pseudocode

DISPLAY

- Displaying information is one of the six basic operations that a computer can perform
- We do this in Banana operation with DISPLAY
- Use this to print something to the console, e.g.

```
display "Please enter your age"
get age
display age
```

- (What's the difference between the two uses of display here? What will each of them print to the console?)

GET

- The ability for a computer to receive information – either from a human being or some other source – is another basic operation
- In Banana, we can use the verb **`get`**:

```
display "Please enter your age"
get age
display age
```

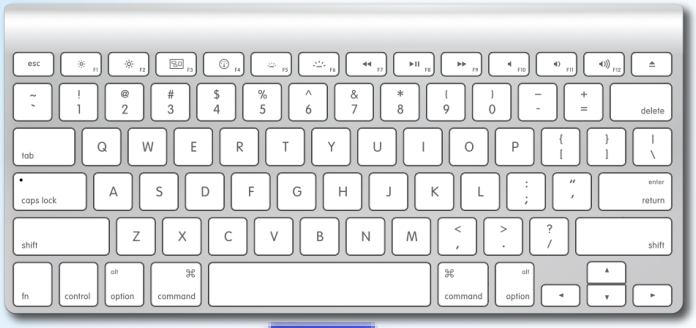
Variables

- In the previous example, age was a *variable*
- Reminder:
  - A variable is like a box that can contain one, and only one thing
  - We can store things in variables and then use them again later on
  - Each "box" has a label on the side – the *variable name*
  - When we want to refer to a variable in our program, we use the "label" of the "box" – i.e. our variable name
  - When we said **`get age`**, we meant, *get input from the user and put it into the box labeled 'age'*
  - When we said **`display age`**, we meant, *display the contents of the box with the label 'age'*.

Variables in Banana

get age

Variables in Banana

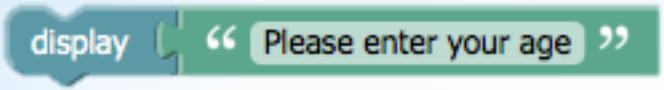


get age

display age

Literal values v variable

- There will also be times we want to use a fixed value in our programs
- When we said
  - display "Please enter your age"
- the text Please enter your age was a *literal value* rather than a variable
  - A literal value is fixed and doesn't change
  - This particular literal value was a *string* – a text value
  - We know that this was a string literal, rather than a reference to a variable named Please enter your age because it was encased in quotes
- (if this is still confusing, think back to the blocks:



- Note the colours – the greenish blue was a literal, the purpleish pink a variable)

SET

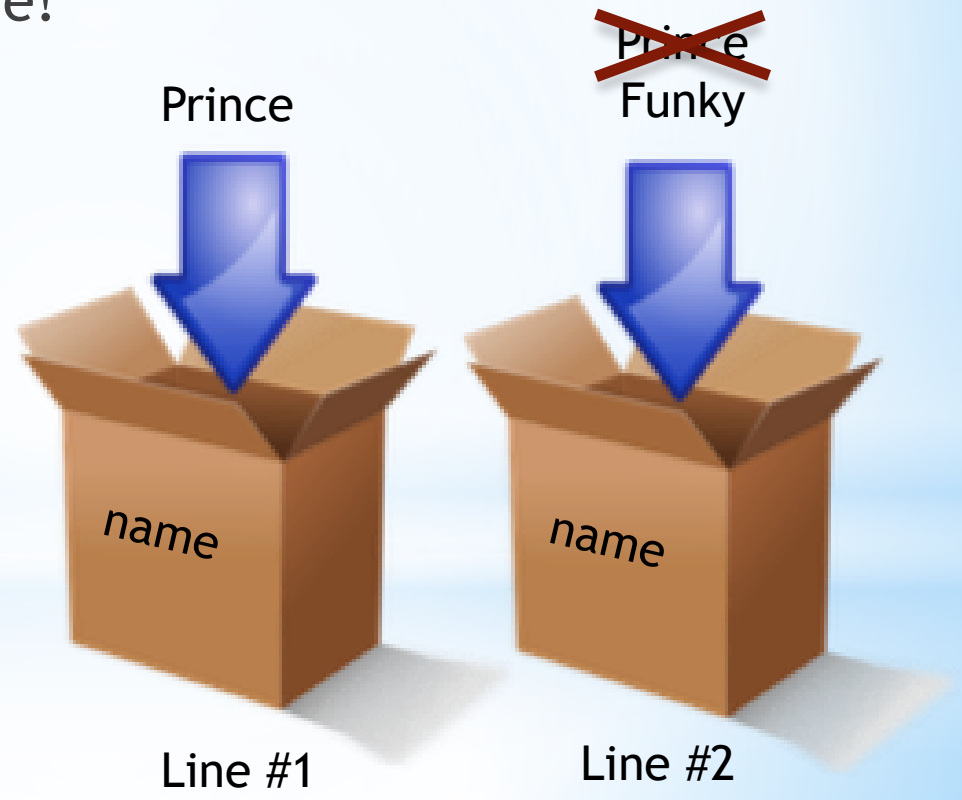
- A computer can store data in memory
  - …or it can "put stuff in labeled boxes"
- To assign a value to a variable in Banana, we use **`set`**:
  - **`set name = "Paul"`**
  - Note the use of the symbol =
    - The left hand side of the = indicates which variable is going to have a value assigned
    - The right hand side of the = indicates what value is going to be put into the variable
  - You can leave the set off if you like
    - **`age = 39`**

Reminder: Variables contain only ONE value at a time

- If you assign a value to a variable, any existing value is replaced with the new one!

1. set name = "Prince"
2. name = "Funky"

- Tip: Get into the habit of using set when you use a variable for the first time, and leaving it off on subsequent uses. That way, when you look back at your pseudocode, you know whether a given line of code is using a new variable or an existing one. This also matches how things work in many programming languages.



Assigning the value of a variable v assigning a literal

- What's the difference between
  - set name = "Paul"
  - set name = Paul
- Both are *potentially* valid!

Assigning the value of a variable v assigning a literal

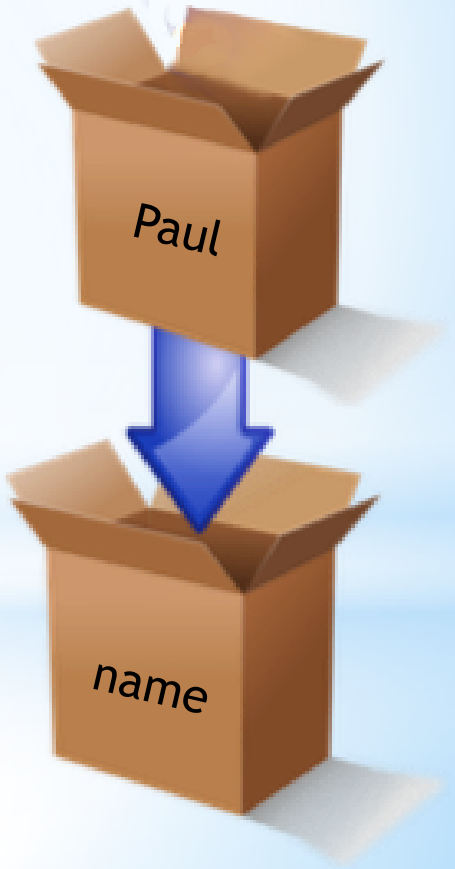
- What's the difference between
  - set name = "Paul"
  - set name = Paul
- Both are *potentially* valid!

- "Take the text Paul and put it into the variable name"

Assigning the value of a variable v assigning a literal

- What's the difference between
  - set name = "Paul"
  - set name = Paul
- Both are *potentially* valid!

- "Take a copy of the content of the variable Paul and put it into the variable name"
  - Note that whatever was in Paul is left unchanged! This would COPY the contents rather than MOVE the contents of Paul



Assigning the value of a variable v assigning a literal

- What's the difference between
  - set name = "Paul"
  - set name = Paul
- Both are *potentially* valid!

What would be printed if we ran the following Banana code?

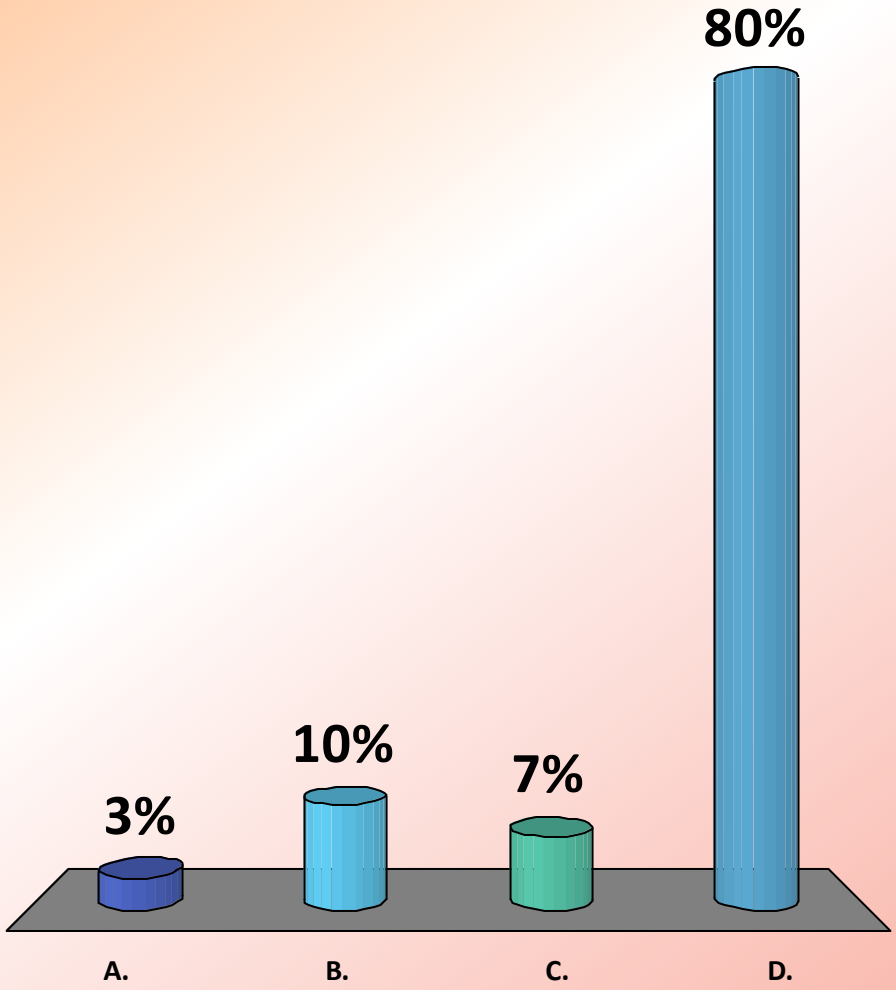
```
set foo = 12
set bar = 14
display "foo"
display bar
```

A. 12
   14

B. 12
   bar

C. foo
   bar

D. foo
   14



Maths operations

- A computer can perform arithmetic
  - Most programs require the computer to perform some sort of mathematical calculation, or formula
  - To be consistent with high-level programming languages, Banana uses the following symbols:

    `+` for Add `-` for Subtract

    `*` for Multiply `/` for Divide

    `%` for Modulo (remainder)
  - When writing mathematical calculations for the computer, standard mathematical 'order of operations' is assumed in Banana and applies in most other programming languages
    - We can use parentheses or brackets `( )` to change the order of operations

Using the arithmetic operators

- display 4+2
  - Results in 6 being printed to the screen
- display 4-2
  - Results in 2 being printed to the screen
- display 4*2
  - Results in 8 being printed to the screen
- display 4/2
  - Results in 2 being printed to the screen
- display 4%2
  - Results in 0 being printed to the screen
    - there is no remainder if you divide 4 by 2…
- display 5%2
  - Results in 1 being printed to the screen
    - …there is a remainder of 1 if you divide 5 by 2

The grammar of programming

- When we have a line of pseudocode like
  - display 4*7
- we are actually saying
  - "display the result of 4 times 7"

- Q: What kind of data is the result of 4 times 7?
  - Is it a number? Several numbers? A string of text? An image? More than one of these?

The grammar of programming

- In virtually all programming languages, if you have a piece of code that results in a number, it will "fit" anywhere a a simple number would fit!
  - display 2+2
  - display 4
    - The result of 2+2 is a number (i.e. 4)
    - The result of 4 on its own is a number (i.e. 4!)
    - Therefore, from a programming point of view, both of these are *grammatically identical!* If you have a keyword or command that can be followed by a number, it can just as readily be followed by a calculation that would *result* in a number*!*

The grammar of programming

- Think back to the blocks:

- Think how both the number 4 and the maths block were represented by a single block
- A single block could be clipped anywhere it would fit – even if it contained other blocks inside it
- This is part of the art of programming – being able to visualise different combinations of symbols as single blocks and understanding where they can be used

Variables and calculations

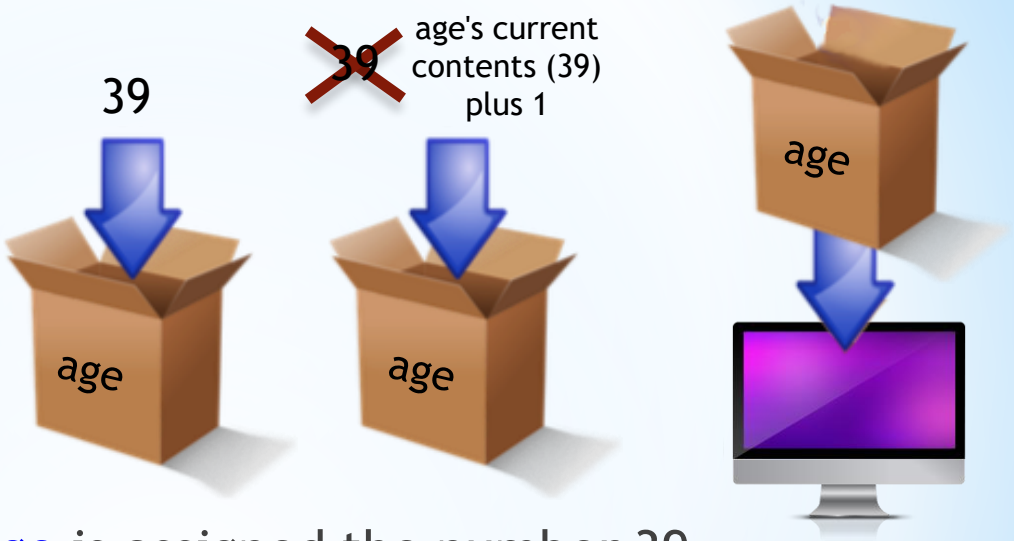
1. set age = 39
2. age = age + 1
3. display age

- displays 40

- **Why?**
  - **Remember the grammar ☺**

Variables and calculations

1. set age = 39
2. age = age + 1
3. display age



- displays 40

- In line 1, the variable age is assigned the number 39
- In line 2…
  - We have a calculation
  - The result of this calculation is a number – i.e. 40
  - This result "fits" into the line of code like any other number
  - This resulting number (40) is placed into age and replaces the existing one (39)
- In line 3, we display the (new) contents of age

More Banana examples

- Birthday

```
display "What is your age?"
get age
age = age + 1
display "On your birthday you will be"
display age
```

```
display "What is your age?"
get age
age = age + 1
display "On your birthday
you will be"
display age
```

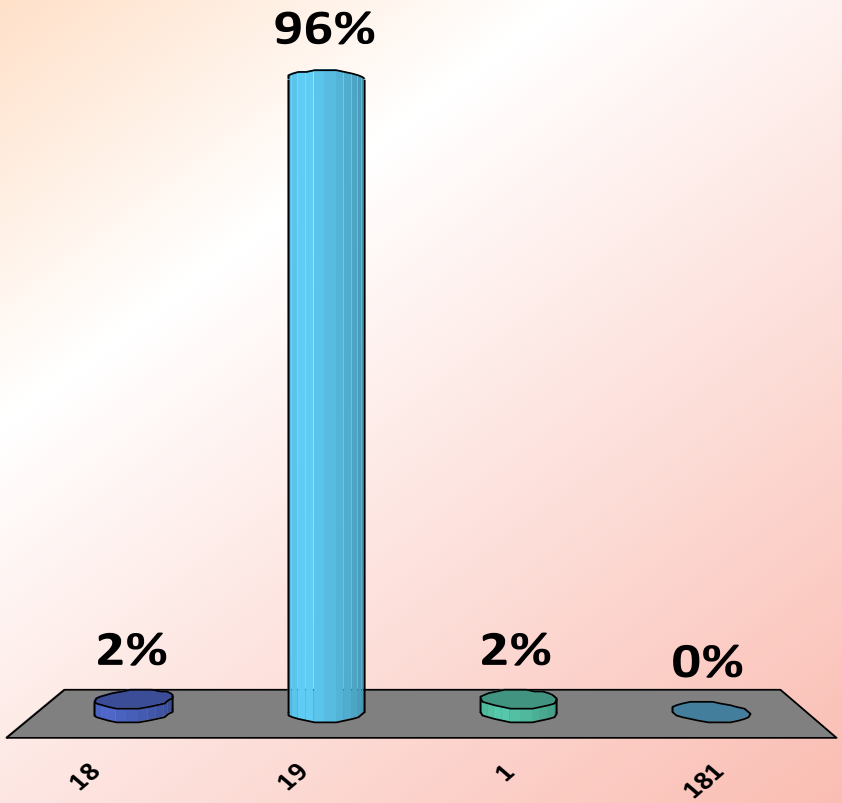
What would this code print if the user typed 18?

A. 18

B. 19

C. 1

D. 181



String variables

- Although we haven't done much of it thus far, you can store text (strings) in your variables

```
set name = "Paul"
set surname = "Neve"
```

- We can also add strings together:

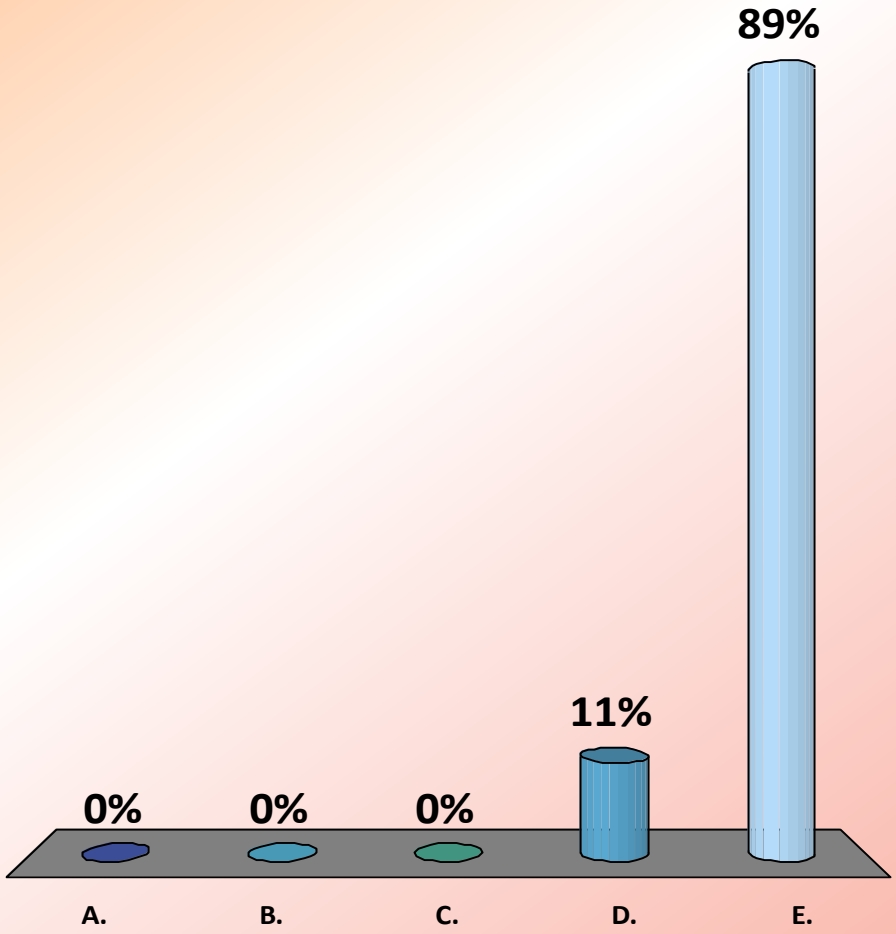
```
set fullname = name+surname
```

- This is called *concatenation*

What would this code print?

```
set name = "Paul"
set surname = "Neve"
set fullname = name+surname
display fullname
```

A. fullname
B. Paul
C. Neve
D. Paul Neve
E. PaulNeve



Concatenation in practice

```
set name = "Paul"
set surname = "Neve"
set fullname = name+surname
```

- Often, when we're dealing with words and concatenating them together, we don't want them to be mashed together but we'll want a space between them
- Spaces should be considered text just like every other symbol – just because we humans can't see them, doesn't mean they're not there
- So we might correct the third line above with the following:

```
set fullname = name+" "+surname
```

Concatentation in practice

```
set fullname = name + " " + surname
```

- If this is confusing, consider this alternative lines of code

```
set fullname = name+"abc"+surname
```

- How is this any different?
  - name is a variable
  - "abc" is a string literal
  - surname is a variable
  - the result is the three of them jointed together

- So, just because what's between the quotes is a space, doesn't make any different – it's still a string literal

Banana examples

- Birthday, refined

```
display "What is your age?"
get age
age = age + 1
display "On your birthday you will be "+age
```

- We can concatenate numbers onto the end of a string, too
- We will often use this to make things display on a single line

- **Remember the grammar mk2** (or, "not just numbers!")
  - If you have a "calculation" that results in a string of text, that piece of code can be used anywhere a string of text would "fit"

Banana examples

- Birthday, refined

```
display "What is your age?"
get age
age = age + 1
display "On your birthday you will be " + age
```

- We can concatenate numbers onto the end of a string, too
- We will often use this to make things display on a single line

- **Remember the grammar mk2** (or, "not just numbers!")
  - If you have a "calculation" that results in a string of text, that piece of code can be used anywhere a string of text would "fit"

More Banana examples

- Calculator

```
display "Enter the first number"
get num1
display "Enter the second number"
get num2
set num3 = num1 + num2
display "The answer is "+num3
```

Making decisions in Banana

- A computer can compare two variables and select one or two alternate actions
  - An important computer operation available to the programmer is the ability to compare two variables and then, as a result of the comparison, select one of two alternate actions
  - To represent this operation in Banana, special keywords are used: IF, ELSE and ENDIF
  - Statements like these that can make decisions are called *conditional statements*

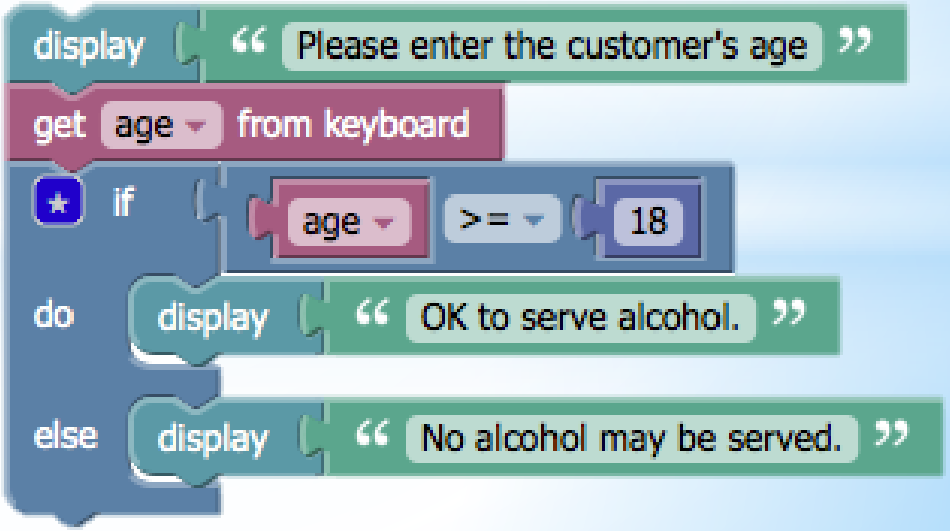
Relational Operators

Relational operators are the symbols used in the condition to be evaluated in `If` statements:

`==`	is equal to (the comparison operator)
`!=`	is not the same as (not equal to)
`<`	less than
`>`	greater than
`<=`	less than or equal to
`>=`	greater than or equal to

Using IF ELSE ENDIF in Banana

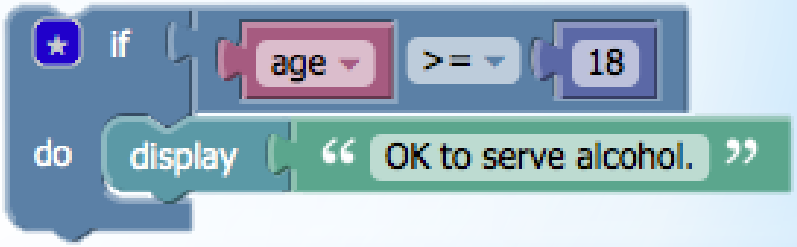
```
display "Please enter the customer's age."
get age
if age >= 18
        display "OK to serve alcohol."
else
        display "No alcohol may be served."
endif
```



The "blocks" of IF statements

- You should think of the IF and ENDIF as being a single block:
  - The IF statement is the start of the block
  - The ENDIF statement is the end of the block
  - Statements go between the IF and ENDIF

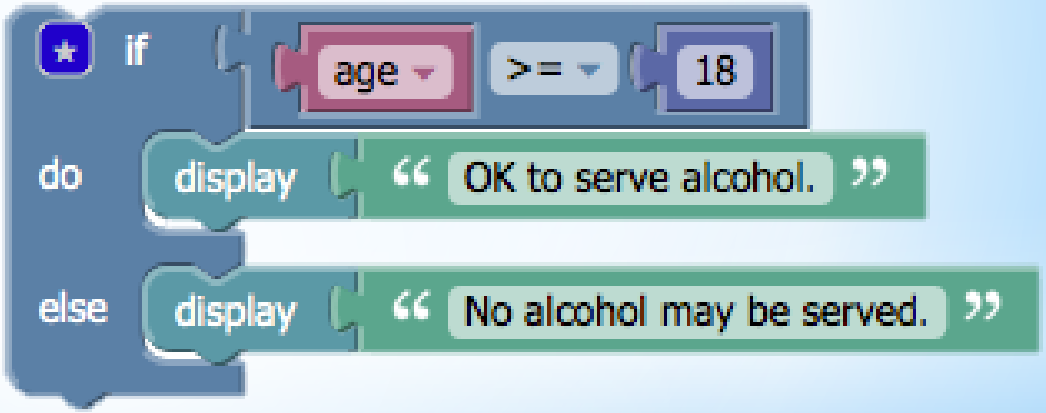
```
if age >= 18
    display "OK to serve alcohol."
endif
```



The "blocks" of IF statements

- If you have an ELSE, this adds another place where statements can go – but the IF/ENDIF combination still represents a single block of program code:

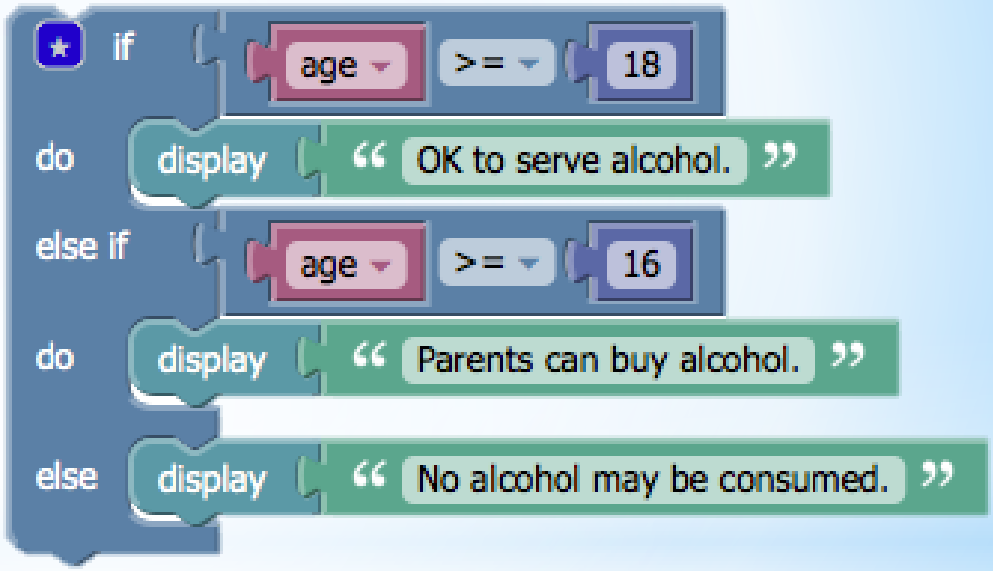
```
if age >= 18
    display "OK to serve …"
else
    display "No alcohol … "
endif
```



The "blocks" of IF statements

- And similarly with IF / ELSEIF / ENDIF:

```
if age >= 18
    display "OK to serve ..."
elseif age >= 16
   display "Parents can buy ... "
else
   display "No alcohol ..."
endif
```



A

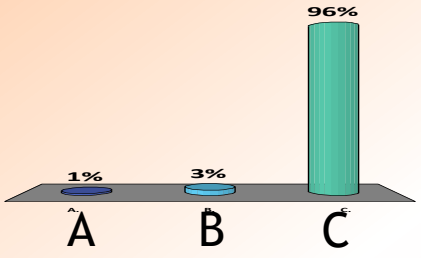
```
get username
if username == "Paul"
  display "Hello, master!"
endif
if username == "Bill"
  display "You owe me a fiver!"
endif
if (username != "Fred")
  display "You are banned!"
else
  display "Welcome!"
endif
```

B

```
get username
if username == "Paul"
  display "Hello, master!"
endif username == "Bill"
  display "You owe me a fiver!"
endif username != "Fred"
  display "You are banned!"
endif
  display "Welcome!"
endif
```

C

```
get username
if username == "Paul"
  display "Hello, master!"
elseif username == "Bill"
  display "You owe me a fiver!"
elseif username != "Fred"
  display "You are banned!"
else
  display "Welcome!"
endif
```



Which of the Banana programs is the correct implementation of the blocks?

Comparison *vs.* Assignment Operators

There is a significant difference between the use of an equals sign (=) as the assignment operator and a double equals sign (==) as the comparison operator.

As an **assignment operator**, the equals sign sets the value of an expression on the right side to the variable on the left side.

As a **comparison operator**, the double equals sign asks the question, "Is the value of the variable on the left side the same as the value of the expression, number, or variable on the right side?"

- a single equals sign (**=**) signifies the assignment operator
- a double equals sign (**==**) signifies the comparison operator

☺ Banana will forgive you getting these wrong but in most programming languages you will be setting sail for fail if you mix these up!

Logical Operators

- Logical operators are used to connect simple conditions into a more complex condition called a **compound condition**.
- The simple conditions each contain one relational operator.
- Using compound conditions reduces the amount of code that must be written.

Combining Logical and Relational Operators to Create Compound Conditions

This code is equivalent to → → →

```
Get X
If  X < 5
   Display "OK"
EndIf
If X > 10
   Display "OK"
EndIf
```

this code. But this code is shorter!

```
Get X
If X < 5 OR X > 10
   Display "OK"
EndIf
```

The AND Operator

- A compound condition consisting of two simple conditions joined by an **AND** is true only if both simple conditions are true. It is false if even one of the conditions is false. The statement:

```
if X > 5 AND X < 10
```

is true only if **X** is 6, 7, 8, or 9. It has to be both greater than 5 and less than 10 at the same time.

- In Banana, you can use the word **AND** or you can use the symbol **&&**
- **&&** is used in most other programming languages – we recommend you get in the habit of using that

The OR Operator

- A compound condition consisting of two simple conditions joined by an **OR** is true if even one of the simple conditions is true. It is false only if both are false. For example:

```
If Response ==“Y” OR Response ==“y”
```

- This is true if **Response** is uppercase or lower case y. For the above condition to be false, **Response** would have to be something other than either ‘Y’ or ‘y’.
- You can also use the symbol || (two pipe characters – look next to the left shift key on a British keyboard)
- Most programming languages use || so try to get in the habit of using that in your own code

The NOT operator

- The NOT operator flips a boolean value - so if it's true, it makes it false; if it's false, it makes it true

```
NOT A < B
```

is true only if **B** is greater than or equal to **A**.

```
if X > 100 AND NOT X == Y
```

is true only if **X** is greater than 100 but not equal to the value of **Y**.

- The more common symbol for NOT in most programming languages is the exclamation mark, !
  - E.g. `if X > 100 AND ! X == Y`
  - Try to get in the habit of using this rather than the full word NOT in your own programs, if possible..

Using brackets with NOT

- A NOT goes with the condition it immediately precedes:

```
if NOT X > 100 AND X == Y
```

  - …will be true if X is *not* greater than 100, *and* X is equal to Y

  - Contrast

```
if NOT (X > 100 AND X == Y)
```

  - How do you think they would differ?
    - Consider if
      - X = 101 Y = 101
      - X = 99 Y = 101
      - X = 99 Y = 99

Loops in Banana

- Banana can repeat a group of actions (a loop)
  - The usual loops that you've seen in your blocks exist:
    - “while…endwhile"
    - " repeat…until"
    - "for…endfor"

Repetition - the for loop

- The for loop is a mainstay of virtually every programming language
- We use a for when we know before the loop starts how many times we need it to repeat

```
for count = 1 to 10
   display "Programming rocks!"
endfor
```

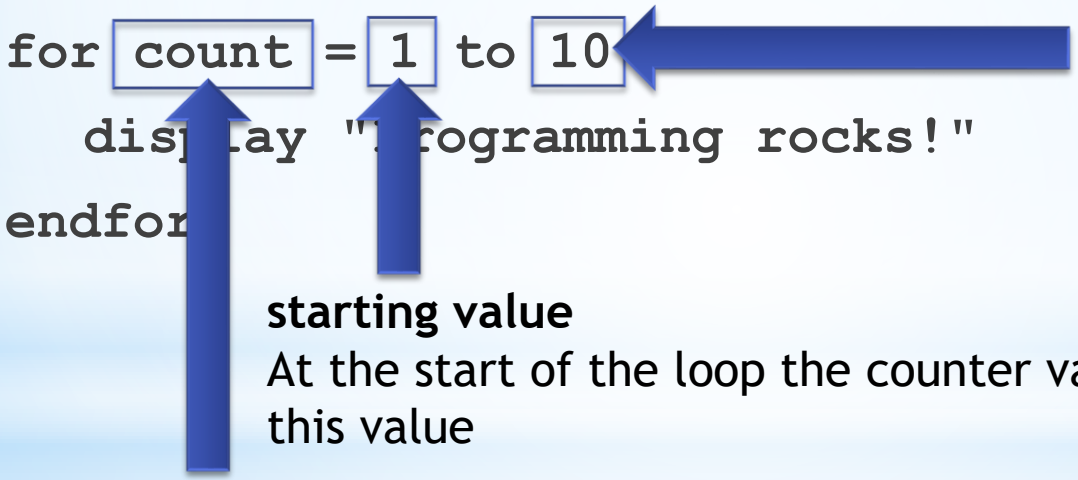
- We could do this without the FOR…
  - …but there would be duplicated code…
  - …what about if we wanted 1000 times?!

This will print

```
Programming rocks!
Programming rocks!
Programming rocks!
Programming rocks!
Programming rocks!
Programming rocks!
Programming rocks!
Programming rocks!
Programming rocks!
Programming rocks!
```

Repetition - the for loop

- The for loop is a mainstay of virtually every programming language
- We use a for when we know before the loop starts how many times we need it to repeat



end value
The loop will end when the counter variable gets to this value

starting value
At the start of the loop the counter variable will contain this value

counter variable
The value in this variable will change every time the loop repeats

Repetition - the for loop

```
display "I'm going to count to 10"
for count = 1 to 10
      display count
endfor
```

- The counter variable is a variable like any other
- It will start at the initial value (1 in this case) and the loop will end when it gets to the end value (10 in this case)

This will print

```
1
2
3
4
5
6
7
8
9
10
```

Repetition - the for loop

```
for banana = 1 to 10
      display banana
endfor
```

- The counter variable is a variable like any other
- It doesn't have to be called **`count`**
- It could be called whatever you like
- You should still follow the rules about sensible variable names, however…
  - …so *don't* call your FOR loop variables **`banana`**! ☺

This will print

```
1
2
3
4
5
6
7
8
9
10
```

Repetition - the for loop

- You can use STEP to specify how much the loop goes up by each time
- If it's left off, it'll assume you mean increase by 1 each time

```
for banana = 1 to 10 step 2
      display banana
endfor
```

This will print

1
3
5
7
9

- Why no 10?

Repetition - the for loop

- Remember to visualise the blocks…


```
for banana = 1 to 10 step 1
        display banana
endfor
```

Repetition - the for loop

```
display "Enter a number…"
get times
for count = 1 to times
   display count
endfor
```

- What would this program do?
  - "Remember your grammar"
  - "Visualise your blocks"

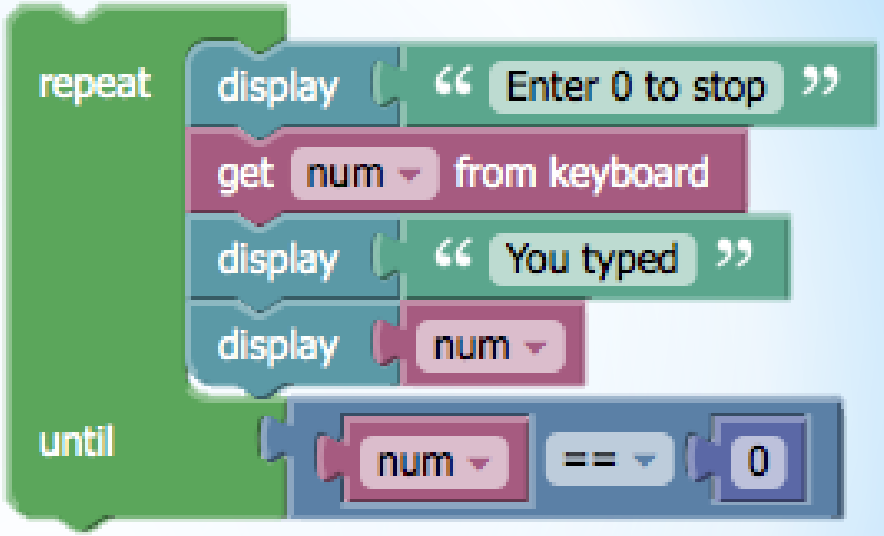
Repetition - the for loop

```
display "Enter a number…"
get times
for count = times to times*2
   display count
endfor
```

- What would this pseudocode do?
  - "Remember your grammar"

Repetition: repeat/until

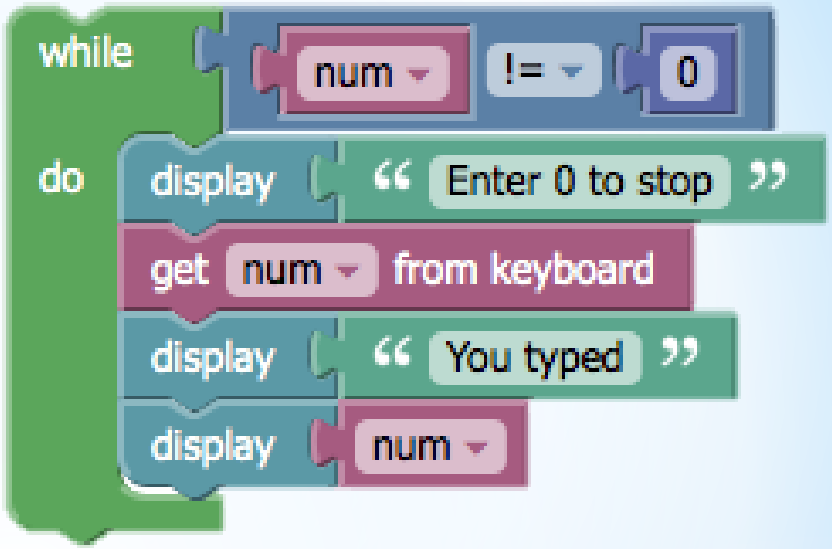
```
repeat
   display "Enter 0 to stop"
   get num
   display "You typed "
   display num
until num == 0
```



- As was the case with the IF statement, the REPEAT/UNTIL construct represents a single block of code
- Try to visualise the blocks when you see text-based code
  - e.g. the comparison in the final line is a single block

Repetition: while/endwhile

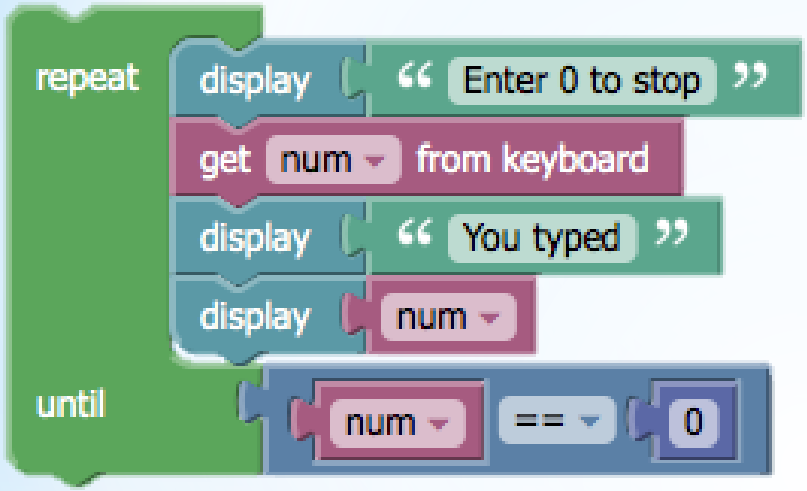
```
while num != 0
   display "Enter 0 to stop"
   get num
   display "You typed "
   display num
endwhile
```

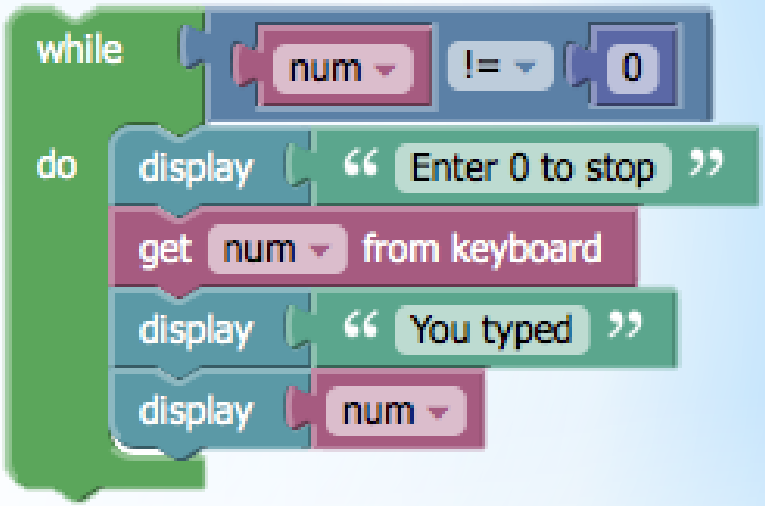


- Again: Visualise the blocks / Remember the grammar!

Repeat v while: a reprise

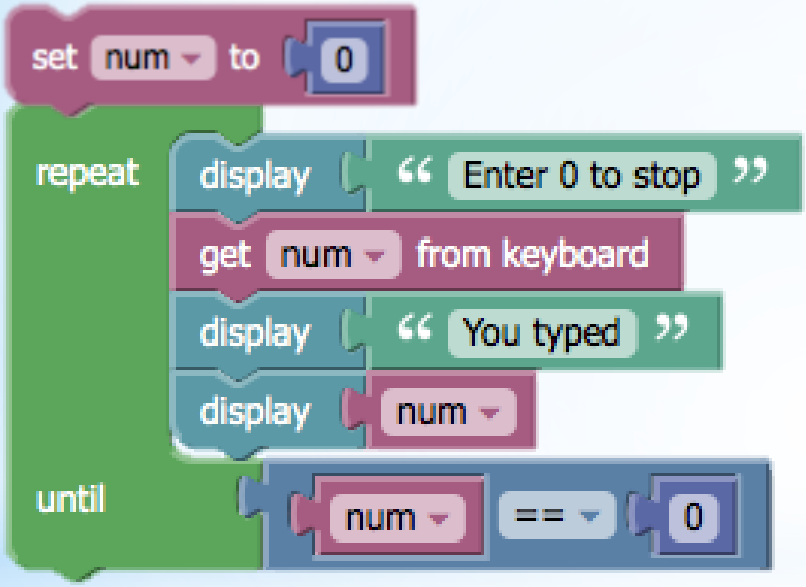
- What's the difference?

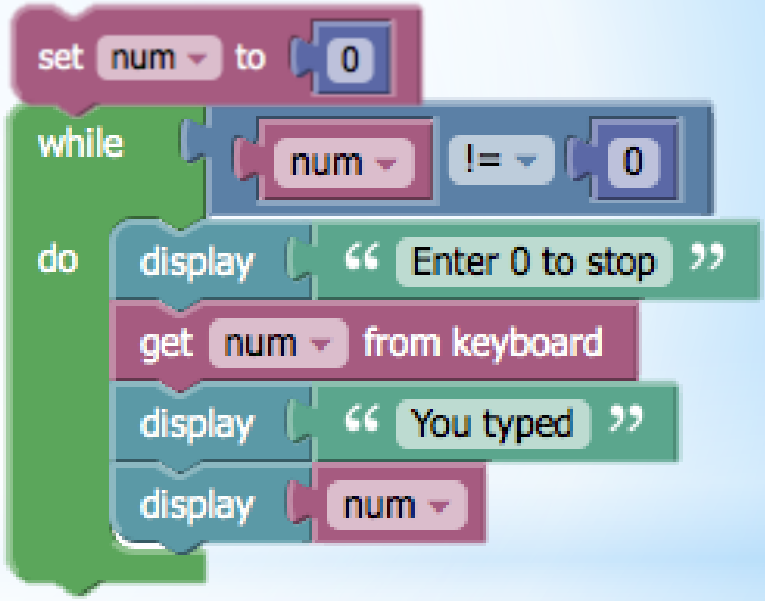




Repeat v while: a reprise

- How about now?





Functions

- Define a function with FUNCTION / ENDFUNCTION

```
function greetUser
   display "Hello there!"
   display "Welcome aboard!"
endfunction
```

- Once again, the boundaries of the code block are defined by the word **`function`** and **`endfunction`**
- Everything in between is the function body
- Call a function with the keyword **`call`**

Functions

```
function func1
   display "I like the number 3"
endfunction

function func2
  display "I like the number 1"
endfunction

function func3
  display "I like the number 2"
endfunction

call func3
call func2
call func1
```

What would be displayed?

Functions and return values

- Specify that your function returns a value using (surprisingly enough) the **`return`** keyword

```
function faveNumber
    return 7
endfunction
```

- Then, to do something with this returned value, use **`resultof`**:

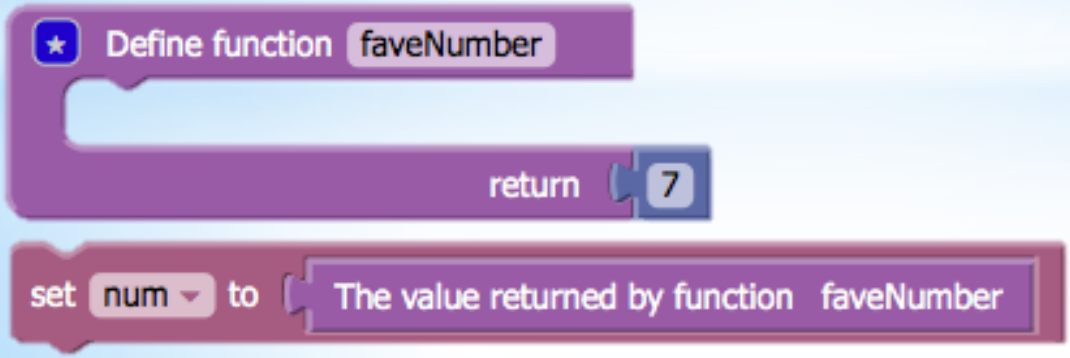
```
set num = resultof faveNumber
```

Functions and return values

```
function faveNumber
   return 7
endfunction

set num = resultof faveNumber
```

- Remember the blocks…
  - A reference to a function that returns a value can be used anywhere that value would fit



Functions and parameters

- If you want a function to take parameters, specify them in brackets after the function's name

```
function addNumbers(numOne,numTwo)
    set result = numOne + numTwo
    display result
endfunction
```

then, for example, to call the function

```
call addNumbers(4,3)
```

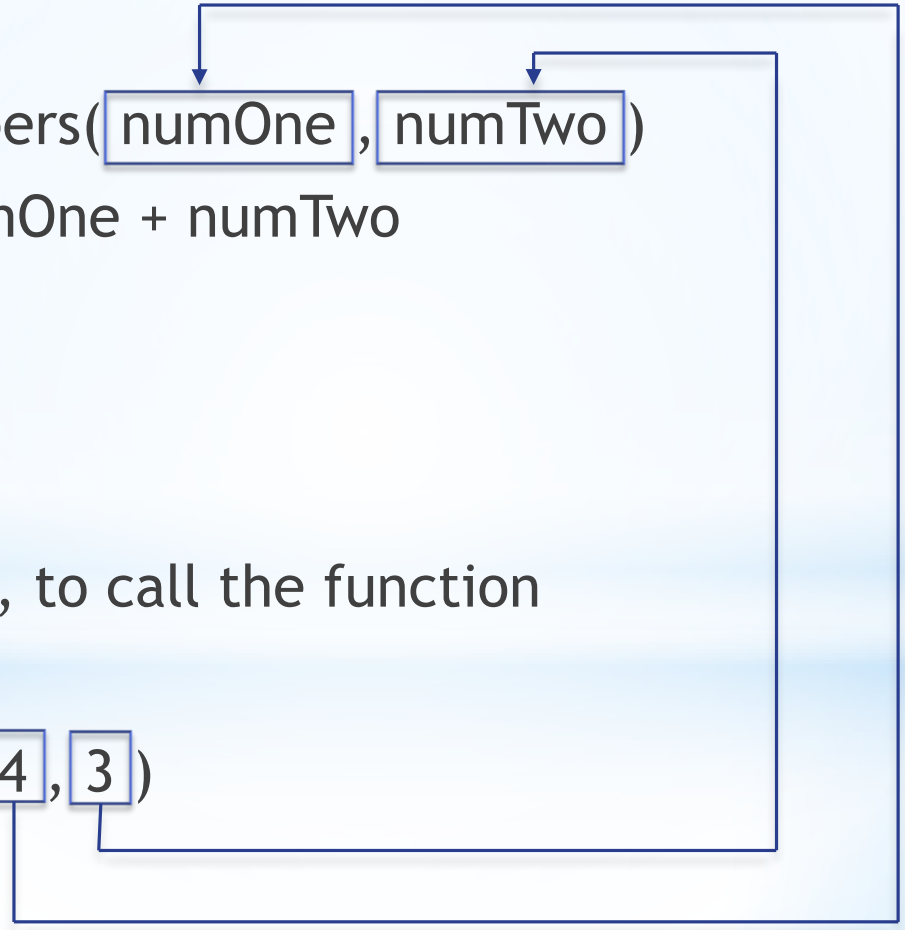
Functions and parameters

- If you want a function to take parameters, specify them in brackets after the function's name

```
function addNumbers( numOne , numTwo )
    set result = numOne + numTwo
    display result
endfunction

then, for example, to call the function

call addNumbers( 4 , 3 )
```



Parameters and return values

```
function addNumbers( numOne , numTwo )
    set result = numOne + numTwo
    return result
endfunction
```

- Use both resultof and the bracketed parameter names:

```
set num = resultof addNumbers(10,20)
```

The end… but

- The following people, please remain at the end:

- K1411737
- K1453038
- K1514279
- K1526857

Summary

- Banana is a pseudocode-like programming language
- So far we've been constructing blocks which get translated into Banana
- Pseudocode is NOT a programming language - just a convention that programmers use and a way of planning out programming solutions before getting involved with a real programming language
- There is no single standard or "correct" version of pseudocode - if you Google around, you will find lots of (conflicting) ways that people write pseudocode
- Pseudocode is more strict than natural language, but less strict than most programming languages
- Banana is designed to be like pseudocode, but you do need to use the correct keywords in the correct place
- But, if you know Banana, you'll be able to read and write pseudocode

Summary

- All of the blocks you've used so far have Banana equivalents
- Some of them map quite nearly onto a single line of code
- Others have keywords that correspond to the start and the end of a block
  - You can put other statements between these start and end statements just as you put blocks inside other blocks

Summary

- A crucial skill of programming is being able to visulise the blocks of your program
- If a combination of symbols, numbers and letters evaluate to a single value (an expression) then that combination will fit anywhere the single value would fit
  - For example:
    - 2+2+4+10+14+1 is 33
    - so 2+2+4+10+14+1 fits anywhere 33 would fit in a program!
- Read the text and try to see the blocks – if you can do that, you'll do OK...! :-)