Flow Control

Prof Matthew Fricke

Version 1.0. Send corrections to mfricke@unm.edu.

Let’s write a python program together that

1. Takes 2 numbers from the user and prints True if the first number is bigger than the second number.

(I'm fine).

2. The program prints False otherwise.

The program should run in script mode – not interactive mode.

Logistics

- I am making the following change to the syllabus:

1) You may now drop 3 lab quiz grades.

Some people have had problems logging in with the lab accounts you were given (cs151-xx). We worked with the computer science support team and checked a few of the accounts. We can’t find any problems. Your lab instructors will help you login.

The story so far…

- You now know the basics of how a computer works (processor and memory)
- We looked at how to:
  - create objects with data in them (e.g. 1, 1.5, (7+2j), “a”, True)
  - (We understood that objects have types (float, bool, int, str, etc.))
  - Convert object types (e.g. float(“1.0”), this is called casting)
  - Assign names to those objects (variables, x = 4)
  - Create objects that are sequences of other objects (lists, tuples, strings)
    (e.g. [“this”, “is”, “a” “list”])
  - Write Python code in interactive mode
  - Write Python programs in script mode that use input() and print()

Enter two numbers and print True if the first is biggest and false otherwise - Line 1

Enter two numbers and print true if the first is larger - Line 2

Enter two numbers and print True if the first is larger - Line 3

Enter two numbers and print True if the first is larger - Line 4

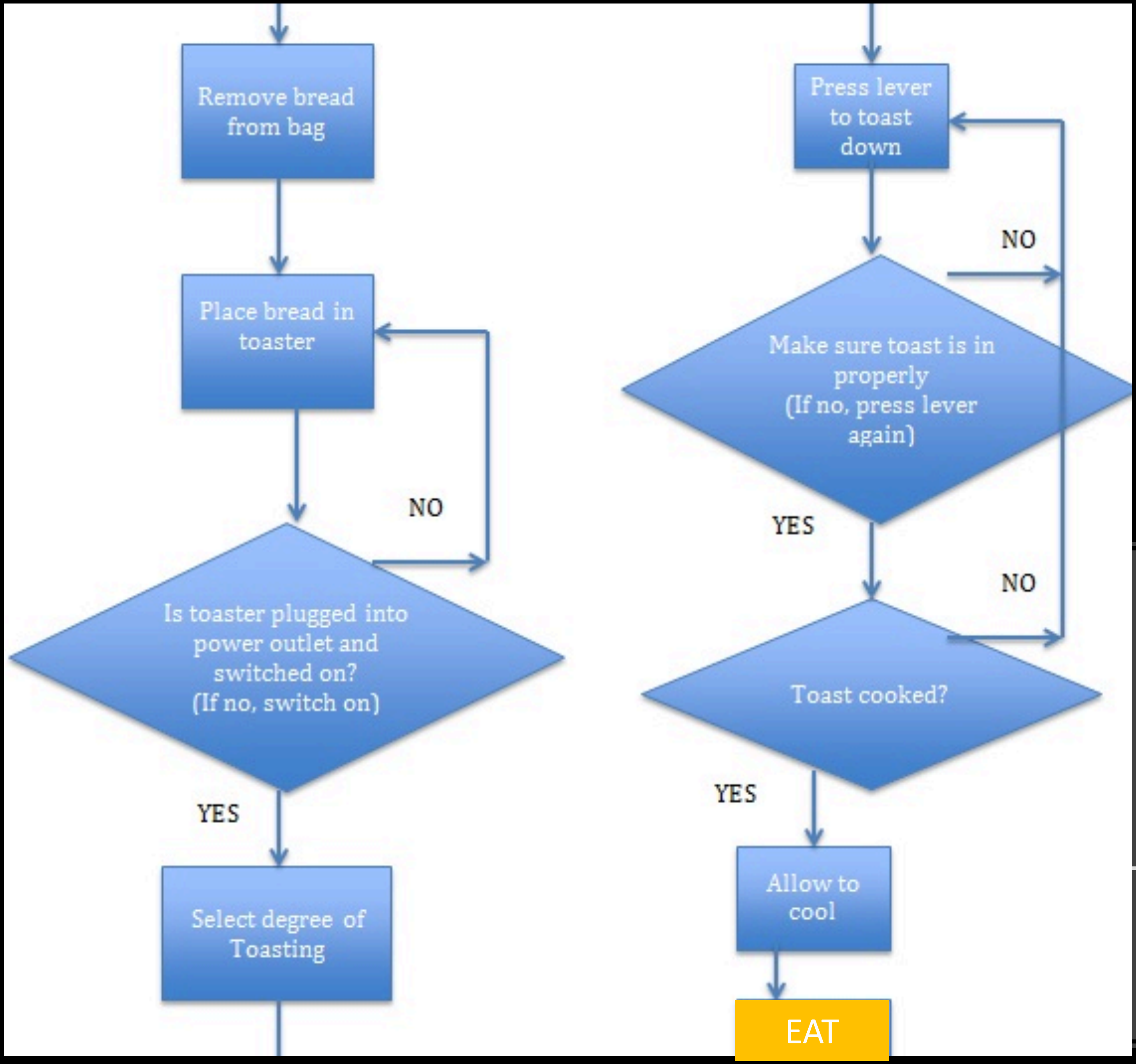
Now we will look at flow control

- Flow control gives your programs the ability to make decisions.
- So far we have used Python as a calculator.

- Flow control enables our programs to become far more powerful.

- In fact adding “if” (if something is true then execute some code, otherwise don’t) means we can write any possible program, execute any possible algorithm. Recall Universal Computer from Lecture 1.

Making Toast



Remove bread from bag

Place bread in toaster

Is toaster plugged into power outlet and switched on? (If no, switch on)

NO

YES

Select degree of Toasting

Press lever to toast down

Make sure toast is in properly (If no, press lever again)

NO

YES

Toast cooked?

NO

YES

Allow to cool

EAT

If…

- We would like our programs to be more than just calculators we want them to make decisions.
- Decision making in programming is called branching. The program goes down one branch if some condition is true and down another if that condition is false.
- Statements that make decisions about what branch of instructions to execute next are called control structures.
- The most common control structure is the if statement.

If…

- Keywords: Programs have “reserved keywords”. These keywords have special meaning.

If is a keyword in Python. You can’t name a variable “if”.

If…

- The syntax of the if control structure is:

```
if boolean_expression:
        statements…
```

- If boolean_expression returns True then the statements inside the braces are executed. If the expression is False then those statements are skipped.

If…

- (green is the interpreter prompt, yellow is the source code, and blue is the output)

```
>>> if 1 == 1:
...       "1 equals 1"
...
'1 equals 1'
```

```
>>> if 1 != 1:
...       "1 is not equal to 1"
...
`1 is not equal to 1'
```

The worst thing about Python

- Indenting…
- Python knows which statements are to be executed if the if condition is true by whether they are indented.

```
>>> if 1 == 1:
...     ■"1 equals 1"
...
'1 equals 1'
```

```
>>> if 1 != 1:
...     ■"1 is not equal to 1"
...
‘1 is not equal to 1'
```

Lines of code with the same indentation are called code blocks

The worst thing about Python

- Indenting...
- Python knows which statements are to be executed if the if condition is true by whether they are indented.

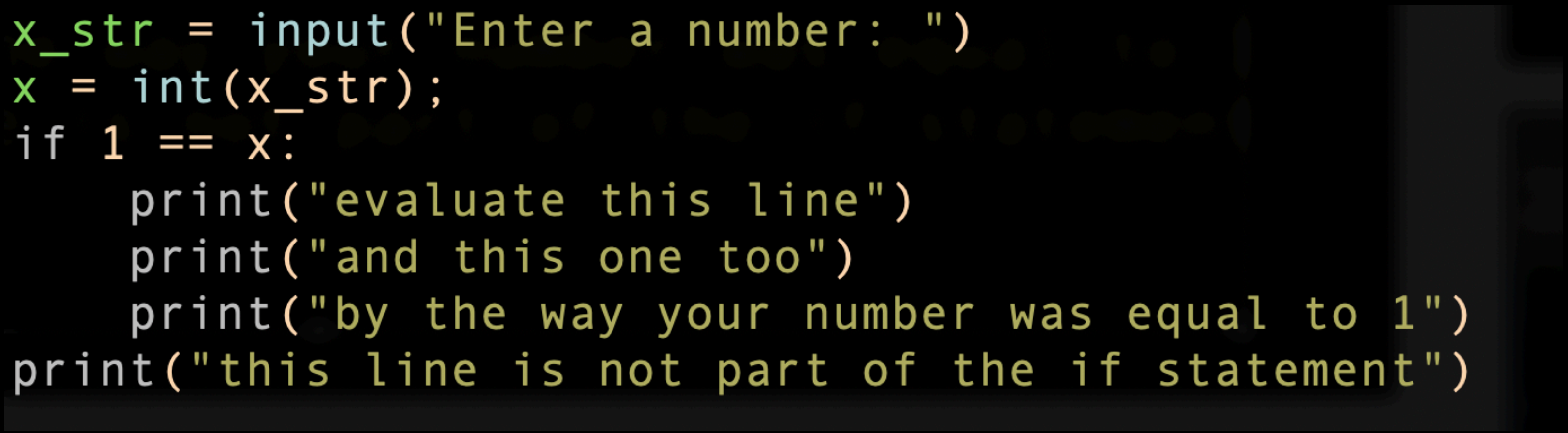
```
>>> if 1 == 1:
...         "true: so evaluate this expression"
...         "true: and this line too"
...
'true: so evaluate this expression'
'true: and this line too'
>>>
```

For now use spaces not tabs.

The amount tabs indent varies too much across editors (including web browsers)

The worst thing about Python

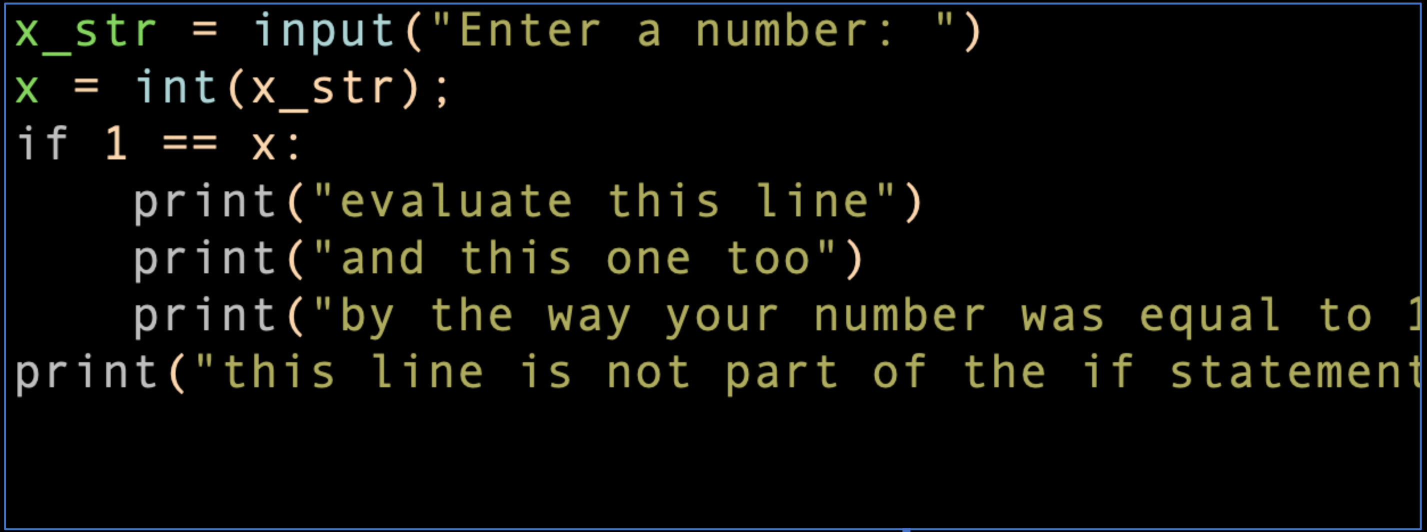
In a file, or ZyBook code space you could write:



```
x_str = input("Enter a number: ")
x = int(x_str);
if 1 == x:
    print("evaluate this line")
    print("and this one too")
    print("by the way your number was equal to 1")
print("this line is not part of the if statement")
```

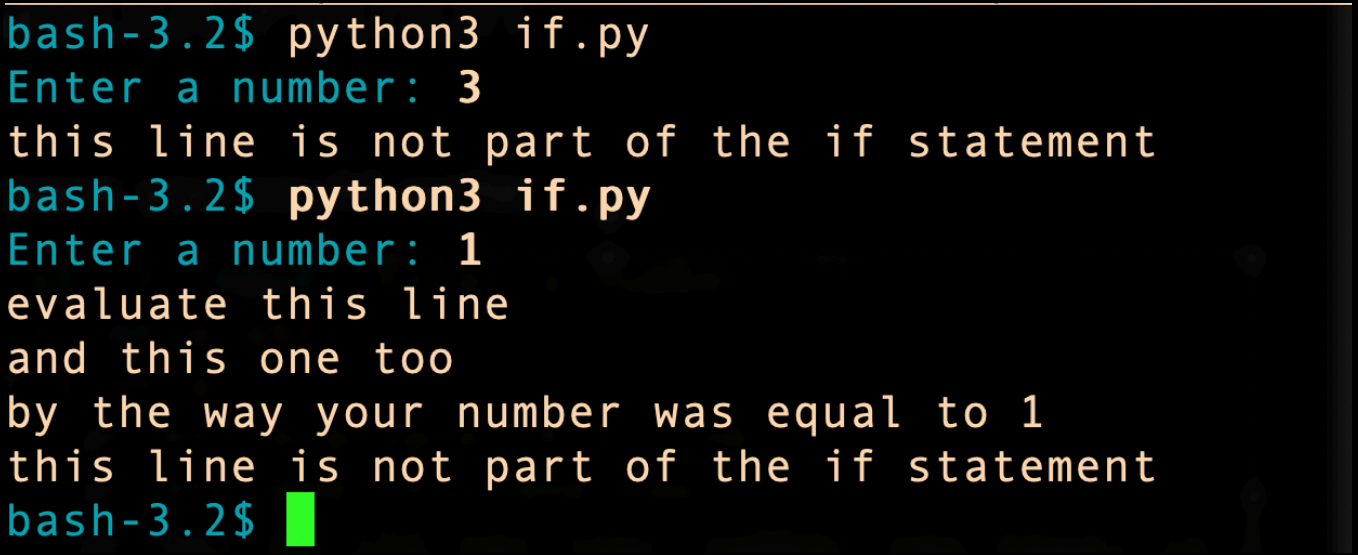
The worst thing about Python

In a file, or ZyBooks you could write:



```
x_str = input("Enter a number: ")
x = int(x_str);
if 1 == x:
    print("evaluate this line")
    print("and this one too")
    print("by the way your number was equal to 1
print("this line is not part of the if statement
```

And then run it to get:



```
bash-3.2$ python3 if.py
Enter a number: 3
this line is not part of the if statement
bash-3.2$ python3 if.py
Enter a number: 1
evaluate this line
and this one too
by the way your number was equal to 1
this line is not part of the if statement
bash-3.2$ 
```

If I saved this code to a file called if.py I could run it on the command line or shell.

Let’s write a python program together that

1. Takes a number from the user.
2. The program prints “Multiple of 3” if that is true.
3. The program prints ”Not a multiple of 3 if that is not true”

The program should run in script mode – not interactive mode.

Check if a number is a multiple of 3 - Line 1 (Section 15)

Check if a number is a multiple of 3 - Line 2 (Section 16)

Check if a number is a multiple of 3 - Line 3 (Section 17)

Check if a number is a multiple of 3 - Line 4 (Section 18)

Check if a number is a multiple of 3 - Line 5 (Section 19)

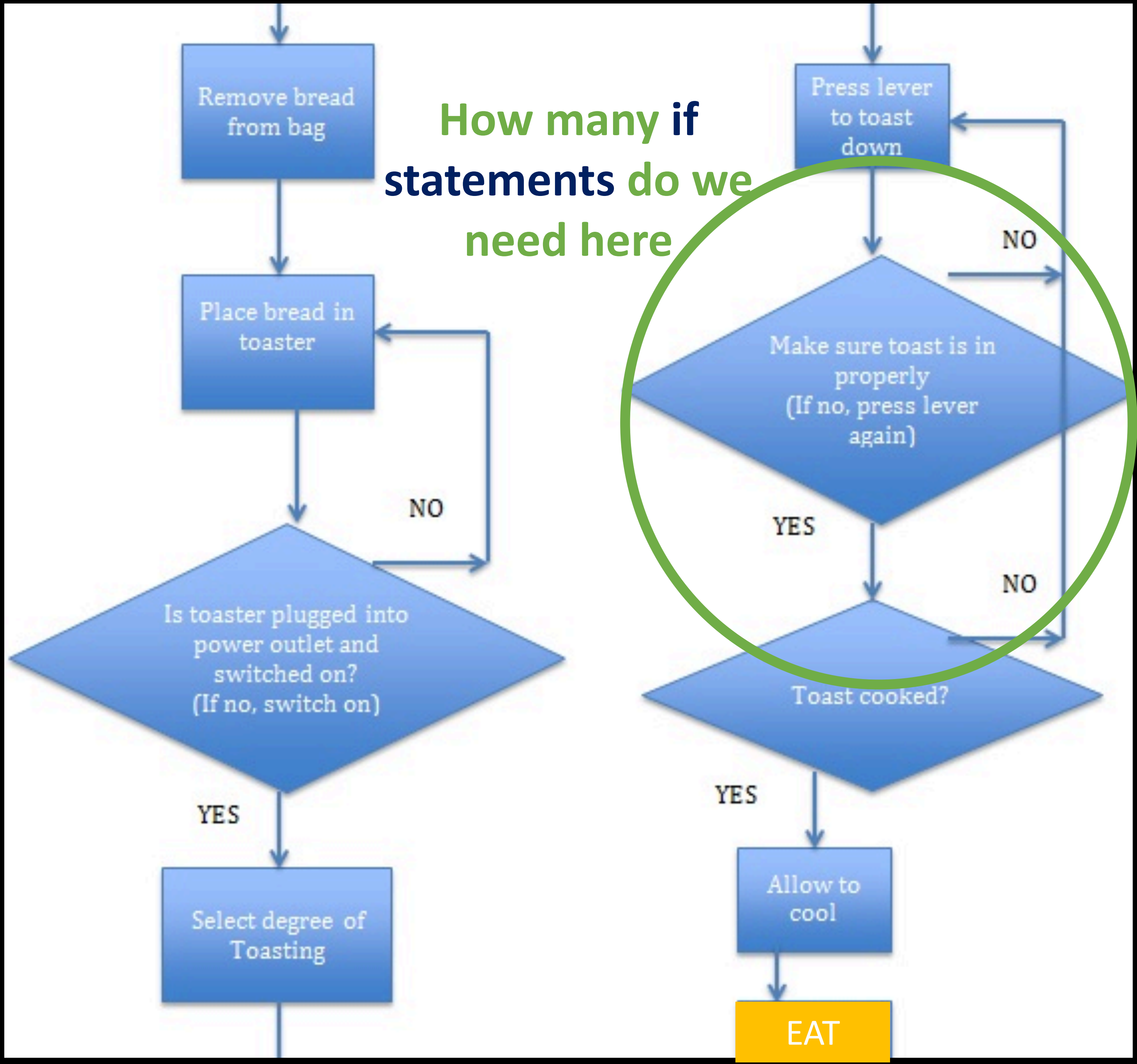
If… elif… else

- Instead of having lots of if statements we can use the easier to understand if… elif… else.

```
if x == 0:
   print(”x equals 0")
elif x == 1:
   print(”x equals 1")
else:
   print(”x isn’t equal to 0 or 1")

print("This statement isn’t part of the If.. Else…"
```

Making Toast

How many **if statements** do we need here



Remove bread from bag

Place bread in toaster

Is toaster plugged into power outlet and switched on? (If no, switch on)

NO

YES

Select degree of Toasting

Press lever to toast down

Make sure toast is in properly (If no, press lever again)

NO

YES

Toast cooked?

NO

YES

Allow to cool

EAT

Iteration

- Instead of having lots of if statements we can use iteration to do the same thing over and over…

```
x = 0
while x < 10:
    x = x + 1
    print(“x equals “ + x)
```

```
while toast != “brown “:
        Press the toast lever

Remove toast
Eat toast
```

Iteration

- Instead of having lots of if statements we can use iteration to do the same thing over and over…

```
x = 0
while x < 10:
    x = x + 1
   print(“x equals “ + x)
```

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

Iteration

- Instead of having lots of if statements we can use iteration to do the same thing over and over…

```
x = 0
while x < 10:
        print(“x equals “ + x)
        x = x + 1
```

```
x equals 1        x equals 0
x equals 2        x equals 1
x equals 3        x equals 2
x equals 4        x equals 3
x equals 5        x equals 4
x equals 6        x equals 5
x equals 7        x equals 6
x equals 8        x equals 7
x equals 9        x equals 8
x equals 10       x equals 9
```

Iteration

- Instead of having lots of if statements we can use iteration to do the same thing over and over…

```
x = 0
while x < 10:
        print(“x equals “ + x)
        x = x + 1
```

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

Iteration

- Instead of having lots of if statements we can use iteration to do the same thing over and over…

```
while x < 10:
    print(“x equals “ + x)
    x = x + 1
```

This can be any Boolean expression.

i.e. any code that returns an object of type bool

```
x equals 0
x equals 1
x equals 2
x equals 3
x equals 4
x equals 5
x equals 6
x equals 7
x equals 8
x equals 9
```

Iteration

- Instead of having lots of if statements we can use iteration to do the same thing over and over…

```
while x < 10:
    print(“x equals “ + x)
    x = x + 1
```

The while loop executes the code block Over and over until the Boolean expression is False.

(So repeats the code as long as the condition it is true.)

This can be any Boolean expression.

i.e. any code that returns an object of type bool

Iteration

- Instead of having lots of if statements we can use iteration to do the same thing over and over…

```
while x < 10:
    print(“x equals “ + x)
    x = x + 1
```

This can be any Boolean expression.

i.e. any code that returns an object of type bool

Common operators that return bool objects:

x > y	is x more than y?
x < y	is x less than y?
x >= y	is x more than or equal y?
x <= y	is x less than or equal y?
x == y	is x equal to y?
x != y	is x no equal to y?
x and y	Are x and y both true?
x or y	Are either x or y true?

Nesting

Code blocks can themselves be, or contain, if statements and iteration.

```
x = 0
while x < 10:
    x = x + 1
    if x % 2 == 0:
        print(“x is even”)
    else:
        print(“x is odd”)

print(“That’s all folks.”)
```

```
1 is odd
2 is even
3 is odd
4 is even
5 is odd
6 is even
7 is odd
8 is even
9 is odd
10 is even
That's all folks.
```

Nesting

Code blocks can themselves be, or contain, if statements and iteration.

```
x = 0
while x < 10:
    x = x + 1
    if x % 2 == 0:
        print(“x is even”)
    else:
        print(“x is odd”)

print(“That’s all folks.”)
```

Nesting

Here is a program that is supposed to print 1 through 5 times 1 through 3. So 1 2 3 4 5 2 4 6 8 10 3 6 9 12 15.

```
x = 0
y = 0
while x < 3:
    x = x + 1
    while y < 5:
        y = y + 1
        print(x*y)
```

Nesting

Here is a program that is supposed to print 1 through 5 times 1 through 3. So 1 2 3 4 5 2 4 6 8 10 3 6 9 12 15.

```
x = 0
y = 0
while x < 3:
    x = x + 1
    while y < 5:
        y = y + 1
        print(x*y)
```

```
1
2
3
4
5
```

Debugging…

Hmmm… this isn’t what I expected… why only one loop?

```
x = 0
y = 0
while x < 3:
    x = x + 1
    while y < 5:
        y = y + 1
        print(x*y)
```

```
1
2
3
4
5
```

Debugging…

Let’s see what is going on…

```
x = 0
y = 0
while x < 3:
    x = x + 1
    while y < 5:
        y = y + 1
        print(x*y)
```

```
1
2
3
4
5
```

Debugging…

Let’s see what is going on…

```
x = 0
y = 0
while x < 3:
    x = x + 1
    print("Outer loop: x is " + str(x))
    print("Outer loop: y is " + str(y))
    while y < 5:
        y = y + 1
        print("    Inner loop: x is " + str(x))
        print("    Inner loop: y is " + str(y))
        print(x*y)
```

Debugging…

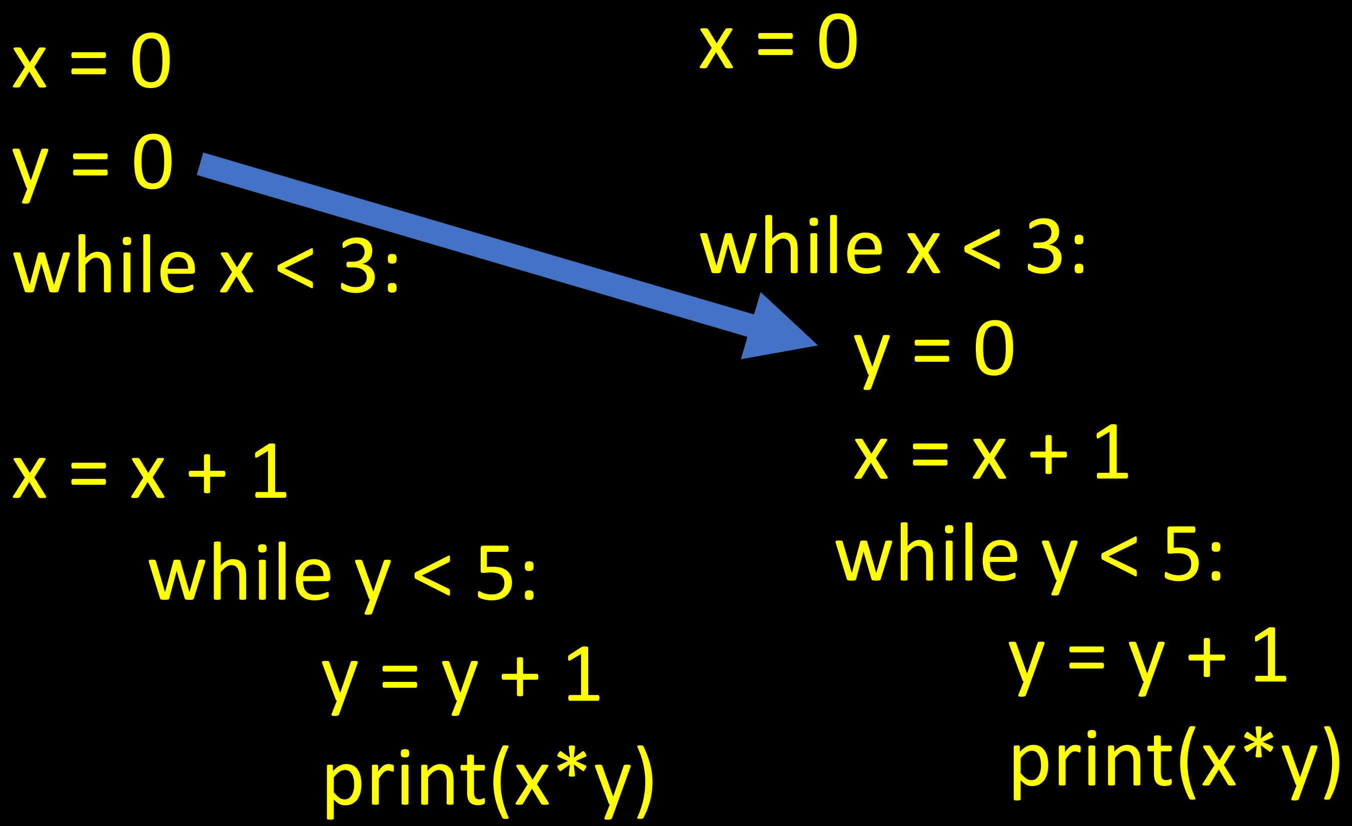
Let’s see what is going on…

```
x = 0
y = 0
while x < 3:
    x = x + 1
    print("Outer loop: x is " + str(x))
    print("Outer loop: y is " + str(y))
    while y < 5:
        y = y + 1
        print("    Inner loop: x is " + str(x))
        print("    Inner loop: y is " + str(y))
        print(x*y)
```

```
Outer loop: x is 1
Outer loop: y is 0
    Inner loop: x is 1
    Inner loop: y is 1
1
    Inner loop: x is 1
    Inner loop: y is 2
2
    Inner loop: x is 1
    Inner loop: y is 3
3
    Inner loop: x is 1
    Inner loop: y is 4
4
    Inner loop: x is 1
    Inner loop: y is 5
5
Outer loop: x is 2
Outer loop: y is 5
Outer loop: x is 3
Outer loop: y is 5
```

Debugging…

Let’s fix it…



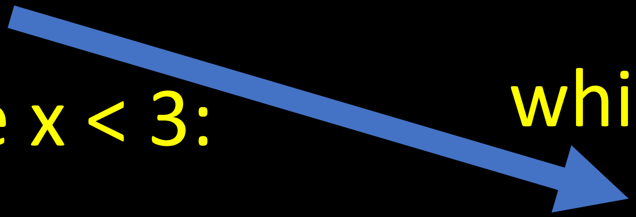
```
x = 0
y = 0
while x < 3:

x = x + 1
    while y < 5:
        y = y + 1
        print(x*y)
```

```
x = 0

while x < 3:
    y = 0
    x = x + 1
    while y < 5:
        y = y + 1
        print(x*y)
```

Debugging…

Let’s fix it…

```
x = 0
y = 0
while x < 3:

x = x + 1
    while y < 5:
        y = y + 1
        print(x*y)
```



```
x = 0

while x < 3:
    y = 0
    x = x + 1
    while y < 5:
        y = y + 1
        print(x*y)
```

```
1
2
3
4
5
2
4
6
8
10
3
6
9
12
15
```

Iteration: For loops

- Instead of having lots of if statements we can use iteration to do the same thing over and over…

```
for <variable> in <sequence>:
        print(i)
```

Iteration: For loops

- Instead of having lots of if statements we can use iteration to do the same thing over and over…

```
for i in [1,2,3,4,5]:
        print(i)
```

Recall that sequence objects include:

Lists

Tuples

Strings

Ranges

Iteration: For loops

- Instead of having lots of if statements we can use iteration to do the same thing over and over…

```
for i in [1,2,3,4,5]:
        print(i)

for i in [“a”,2, “b”, 4, 5]:
        print(i)
```

```
1
2
3
4
a
2
b
4
5
```

Iteration: For loops

- Instead of having lots of if statements we can use iteration to do the same thing over and over…

```
for i in “this is a string”:
        print(i)
```

```
t
h
i
s

i
s

a

s
t
r
i
n
g
```

Iteration: For loops

- Instead of having lots of if statements we can use iteration to do the same thing over and over…

```
for i in “this is a string”:
        print(i)
```

```
t
h
i
s

i
s

a

s
t
r
i
n
g
```

Notice that spaces are also elements of the string just like letters and numbers are.

Iteration: For loops

- Instead of having lots of if statements we can use iteration to do the same thing over and over…

```
for i in range(10, 20, 2):
    print(i)
```

```
10
12
14
16
18
```