Geophysics 501
Computational Methods for the Geosciences

Flow Control and Lists

Zach Zens

Shell versions

- Thompson shell – sh
- Bourne shell
- Bourne again shell - BASH
- TCSH
- Korn Shell - KSH
- Z Shell - ZSH

Shell versions

- Thompson shell – sh
- Bourne shell
- Bourne again shell - **BASH**
- **TCSH**
- Korn Shell - **KSH**
- Z Shell - **ZSH**

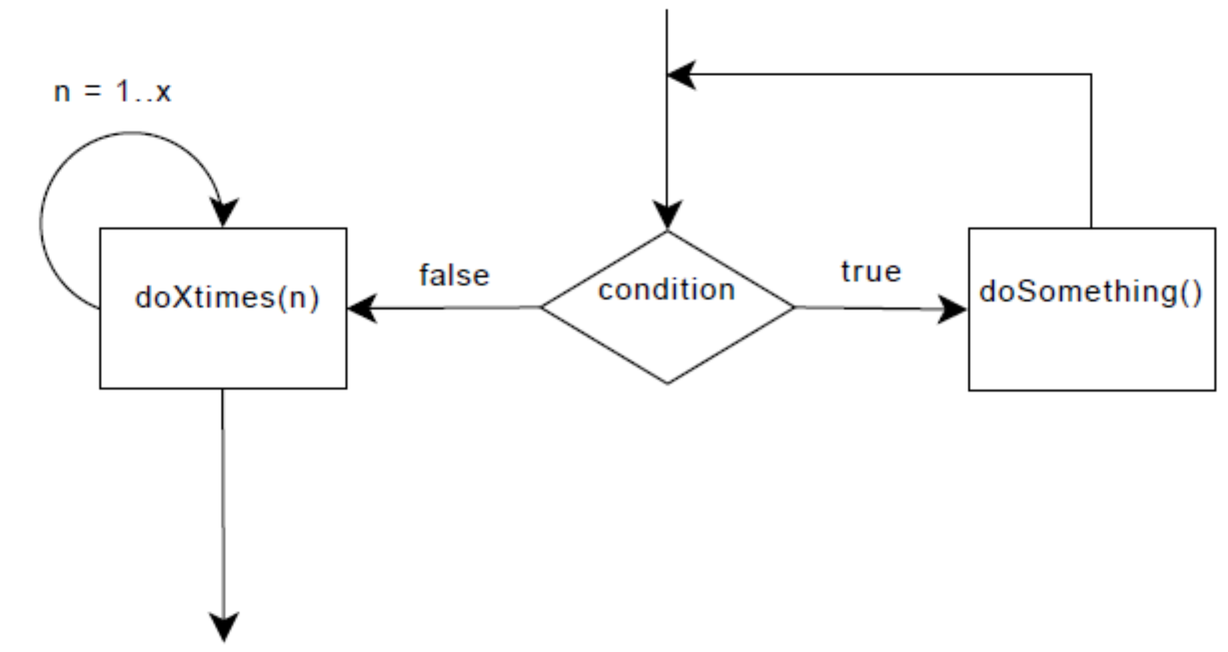
General programming advice

- Your code should be clarified with comments (especially confusing parts)
  - Python and Shell use ‘#’
  - Everything after comments will be ignored
    - print 2 + 2 # This prints 4
    - print 2 # + 2 This prints 2
- Use indentation and white space (Python forces indentation!)
- Use meaningful names for variables
- Decide on one formatting and naming scheme and stick with it

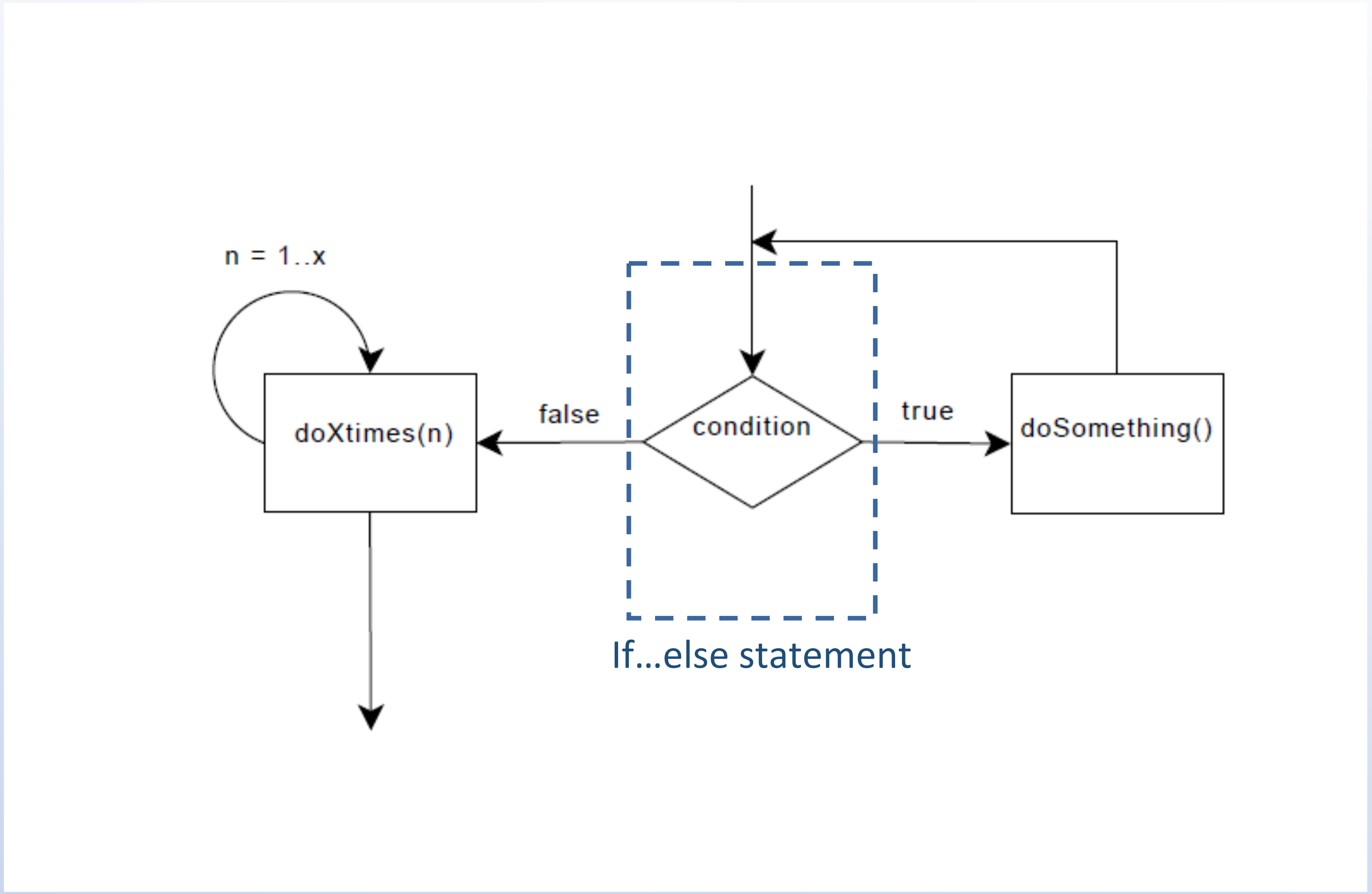
Indentation

- Indentation and white space make your program readable—no one wants to read a giant wall of text
- When using if…else statements or loops in Python, your program will not run if you do not use proper indentation
- Other languages enforce other rules to mark bodies of statements/ loops

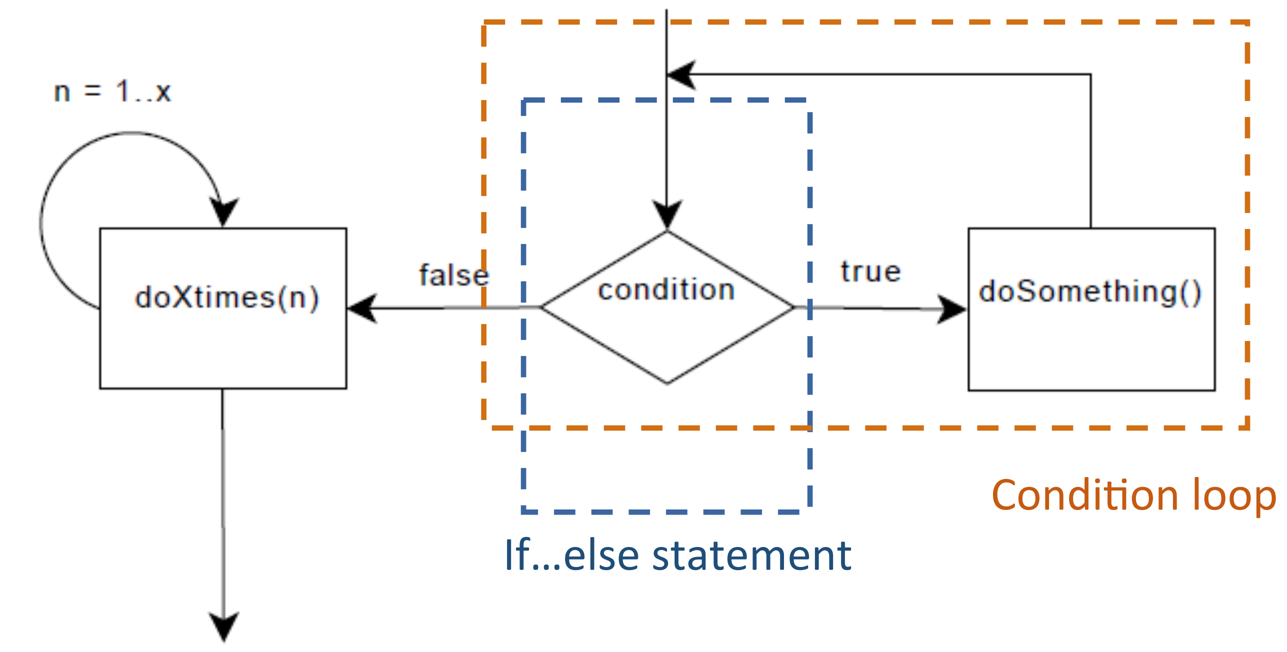
Flow control



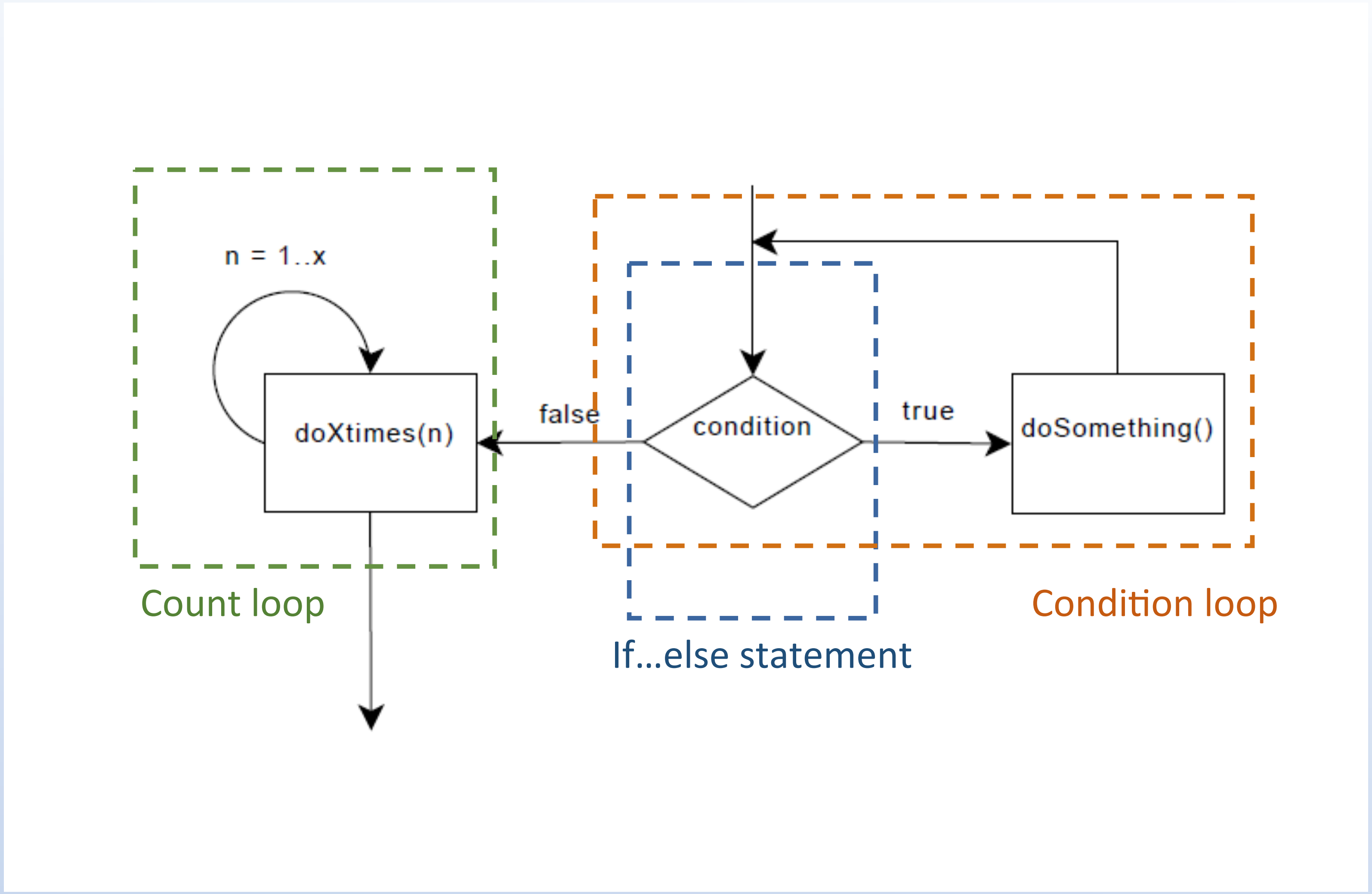
Flow control



Flow control



Flow control



Why do you need flow control?

- Programs without flow control run the same sequence of commands every time the program is run
- With flow control, different statements are run depending on conditions which you define
- Conditions change depending on user input, data, calculations, random variables, etc

Conditionals

- Conditionals are used to see if some condition is true (1) or false (0)
- Conditionals use two types of operators:
  - Relational
  - Logical

Relational operators

Python	**Bash**
• == or ‘is’	• -eq
• != or ‘is not’	• -ne
• >	• -gt
• <	• -lt
• >=	• -ge
• <=	• -le

Logical operators

Python

- &, and
- |, or
- ^
- ~, not

Bash

- -a
- -o
- !

Python structure

```
if <condition>:
        <statement>
elif <condition>:
        <statement>
else:
        <statement>
```

Python structure

```
if <condition>:
        <statement>
elif <condition>:
        <statement>
else:
        <statement>
```

Python structure

```
if <condition>:
        <statement>
elif <condition>:
        <statement>
else:
        <statement>
```

Python example

```
if grade > 90:
        print “A”
elif grade > 80:
        print “B”
elif grade > 70:
        print “C”
elif grade > 60:
        print “D”
else:
        print “F”
```

Bash structure

```
if [ <condition> ]
then
        <statement>
elif [ <condition> ]
then
        <statement>
else
        <statement>
fi
```

Bash structure

```
if [ <condition> ]
then
        <statement>
elif [ <condition> ]
then
        <statement>
else
        <statement>
fi
```

Bash structure

```
if [ <condition> ]
then
        <statement>
elif [ <condition> ]
then
        <statement>
else
        <statement>
fi
```

Bash structure

```
if [ <condition> ]
then
        <statement>
elif [ <condition> ]
then
        <statement>
else
        <statement>
fi
```

Bash example

```
if [ $grade -gt 90 ]; then
		echo “A”
elif [ $grade -gt 80 ]; then
		echo “B”
elif [ $grade -gt 70 ]; then
		echo “C”
elif [ $grade -gt 60 ]; then
		echo “D”
else
		echo “F”
fi
```

Nested if…else example

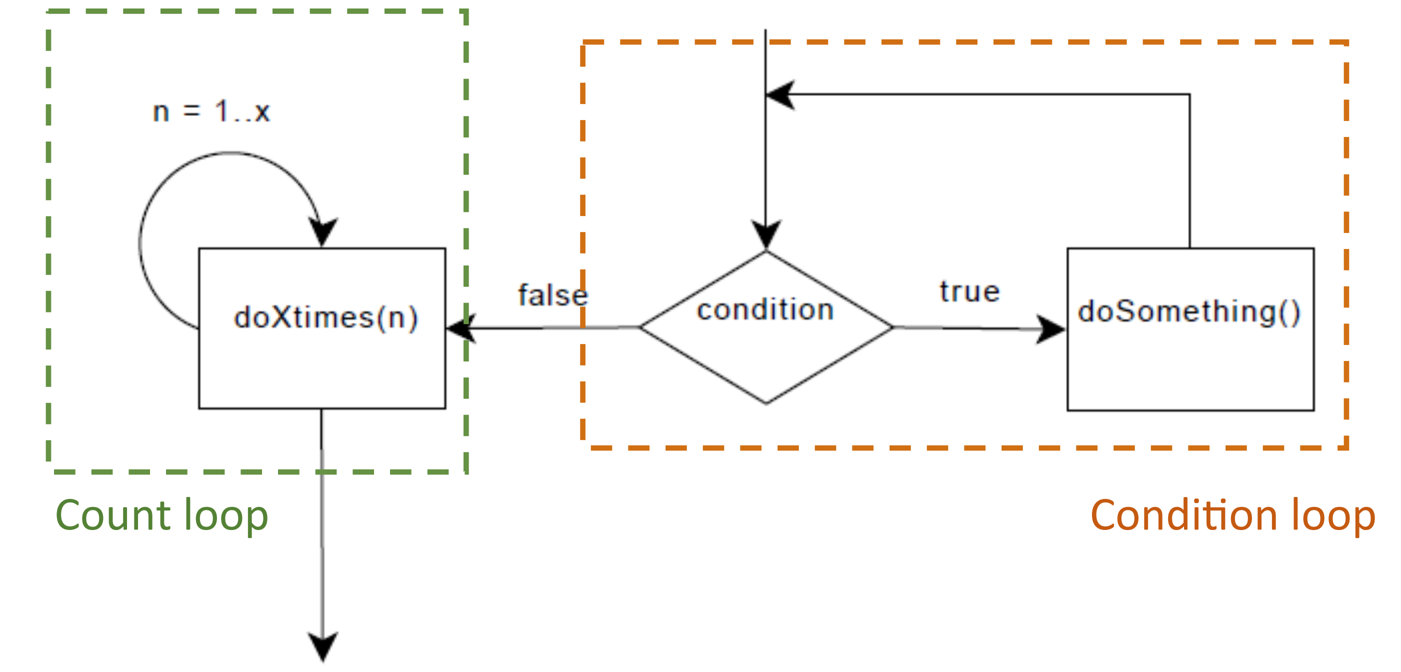
Python

```
if x > 50 and x < 100:
        if y > 30 and y < 40:
                print “Within range”
        elif y > 20 and y < 50:
                print “Almost in range”
```

Bash

```
if [ $x -gt 50 -a $x -lt 100 ]
        if [ $y -gt 30 -a $y < 40 ]
                echo “Within range”
        elif [ $y -gt 20 -a $y -lt 50 ]
                echo “Almost in range”
        fi
fi
```

Loops


n = 1..x
doXtimes(n)
false
condition
true
doSomething()
Count loop
Condition loop

Conditionals vs Loops

- Conditional statements execute statements if certain conditions are met

- Loops execute certain statements over and over until a condition is met.

Types of loops

Python

- While loop
- For loop

Bash

- While loop
- Until loop
- For loop
- Select loop

While/Until loop structure

Python

```
while <condition>:
        <statement>
```

Bash

```
while [ <condition> ]
do
        <statement>
done
```

Bash

```
until [ <condition> ]
do
        <statement>
done
```

While loop example

Python

```
n = 6
while n > 0:
        print n
        if n % 3 == 0:
                print “n is divisible by 3”
        n -= 1
```

Bash

```
n=6
while [ $n -gt 0 ]
do
        echo $n
        if (( $n % 3 == 0 )) ; then
                echo “$n is divisible by 3”
        let n=$n-1
done
```

Output

```
6
6 is divisible by 3
5
4
3
3 is divisible by 3
2
1
```

Infinite loops

```
while True:
        print “Hello”
```

- If the condition is set to something that will always be true, the loop will repeat infinitely
- To get out, you must press Ctrl+C

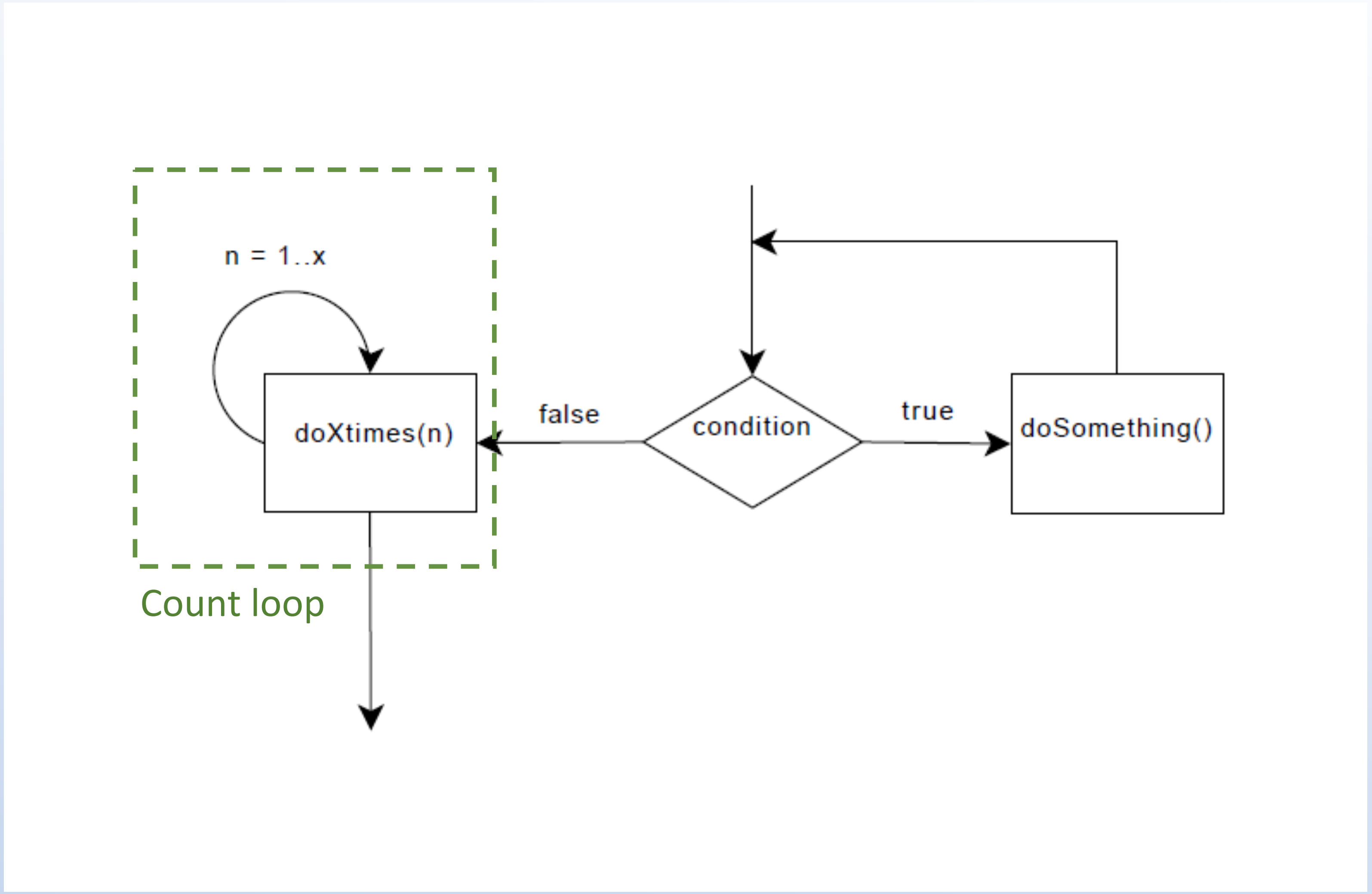
Breaking a loop example (Python)

```
num = 10
while True:
    num = num – 1
    if num == 0:
        break
    if num % 2 == 0:
        print str(num) + " is even"
```

Output:

```
8 is even
6 is even
4 is even
2 is even
```

For Loops


n = 1..x
doXtimes(n)
false
condition
true
doSomething()
Count loop

Sequences

- A sequence is an ordered collection of objects, or elements
- Sequences in Python include strings, lists and tuples
- Lists are very useful sequences which are used for organizing information

[1, 2, 3, 4, 5, 6]
[‘a’, ‘b’, ‘c’, ‘d’]

For loops

Python

```
for <var> in <sequence>:
        <statement>
```

Bash

```
for <var> in <sequence>
do
        <statement>
done
```

For loop example

Python

```
for num in range(0,10):
	print num
```

Bash

```
for num in 0 1 2 3 4 5 6 7 8 9
#same as: for num in `seq 1 10`
do
	print $num
done
```

Output

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

Nested loops example (Python)

```
num = 3
while num > 0:
        print num
        for letter in [‘a’, ‘b’, ‘c’]:
                print letter
        num -= 1
```

Output:

```
3
a
b
c
2
a
b
c
1
a
b
c
```

Lists

- A list can be thought of as a value which contains multiple elements or values.
- Elements can be integers, floats or strings or a combination of all of them.
- One or more elements can be referenced using indexing

Example:

```
Newlist = [45, 67, 34, 80]
```

Indexing

- Each element in a list is assigned a value, starting with 0
- Elements may be accessed by calling its index:
- Negative indices may also be used

```
Newlist[0]
$> 45
Newlist[1]
$> 67
```

```
[45, 67, 34, 80]
  0    1   2   3
```

Indexing

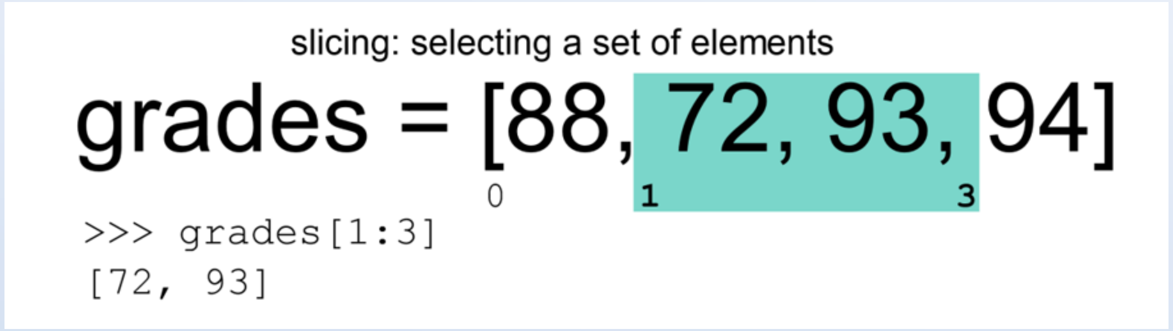
- Each element in a list is assigned a value, starting with 0
- Elements may be accessed by calling its index:
- Negative indices may also be used (Python specific)

```
Newlist[-1]
$> 80
Newlist[-3]
$> 67
```

```
 -4   -3   -2   -1
[45, 67, 34, 80]
  0    1    2    3
```

Slicing

- You can call a sublist by using slicing (Python specific)


slicing: selecting a set of elements
grades = [88, 72, 93, 94]
0
1
3

```
>>> grades[1:3]
[72, 93]
```

Changing indices

- Strings are immutable
- Lists are mutable

```
$> txt = “car”
$> txt[0] = “t”
TypeError: 'str' object does not support item
assignment
```

```
$> newlist = [50, 45, 30]
$> newlist[1] = 40
$> newlist
[50, 40, 30]
```

Nested lists example (Python)

```
nestedlist = [5, 6, [7, 8, 9, 10], 11, [12, 13]]
nestedlist[0]
nestedlist[1]
nestedlist[2]
```

Output:

```
5
6
[7, 8, 9, 10]
```

Accessing subelements

```
nestedlist = [5, 6, [7, 8, 9, 10], 11, [12, 13]]
nestedlist[2][0]
nestedlist[2][1]
nestedlist[2][0:2]
```

Output:

```
7
8
[7, 8]
```

Index out of range (Python)

```
>>> new_list = [1,2,3]      #make new list
>>> new_list[3]             #access last item?

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Oops – read error messages!

Fizzbuzz test

- Simple flow control and iteration program commonly given in interviews
- Allegedly, most graduates fail this test

Rules

- Print a list of numbers from 1 to 100
- For multiples of 3, print “Fizz” instead of the number
- For multiples of 5, print “Buzz” instead of the number
- For multiples of 3 & 5, print “FizzBuzz” instead of the number

More examples

Python

```
num = 0
while true:
        num = int(input “Enter a number from 1-10:”)
        if num > 0 and num < 11:
                break
        print “%num is not a number from 1-10.”
```

Bash

```
num=0
while [ True ]
do
        echo “Enter a number from 1-10:”
        read num
        if [ $num -gt 0 -a $num –lt 11 ]; then
                break
        fi
        echo “$num is not a number from 1-10.”
done
```

Output

```
Enter a number from 1 – 10:
$> 16
16 is not a number from 1 – 10
$> -9
-9 is not a number from 1 – 10
$> 5
```

Another example (Python)

```
sample1 = ['zz-1', 0.31, 14.5, 53.7]  #name, Ti, Al, Zr (ppm)
sample2 = ['zz-2', 0.38, 14.6, 60.5]
newlist = [[], []]
for x in range(0, len(sample1)):
    if type(sample1[x]) is not str and float(sample1[x]) > 20:
        sample1[x] /= 10000
        sample2[x] /= 10000
    newlist[0].append(sample1[x])
    newlist[1].append(sample2[x])

print newlist
```

Output:

```
[['zz-1', 0.31, 14.5,
0.005370000000000001], ['zz-2',
0.38, 14.6, 0.00605]]
```