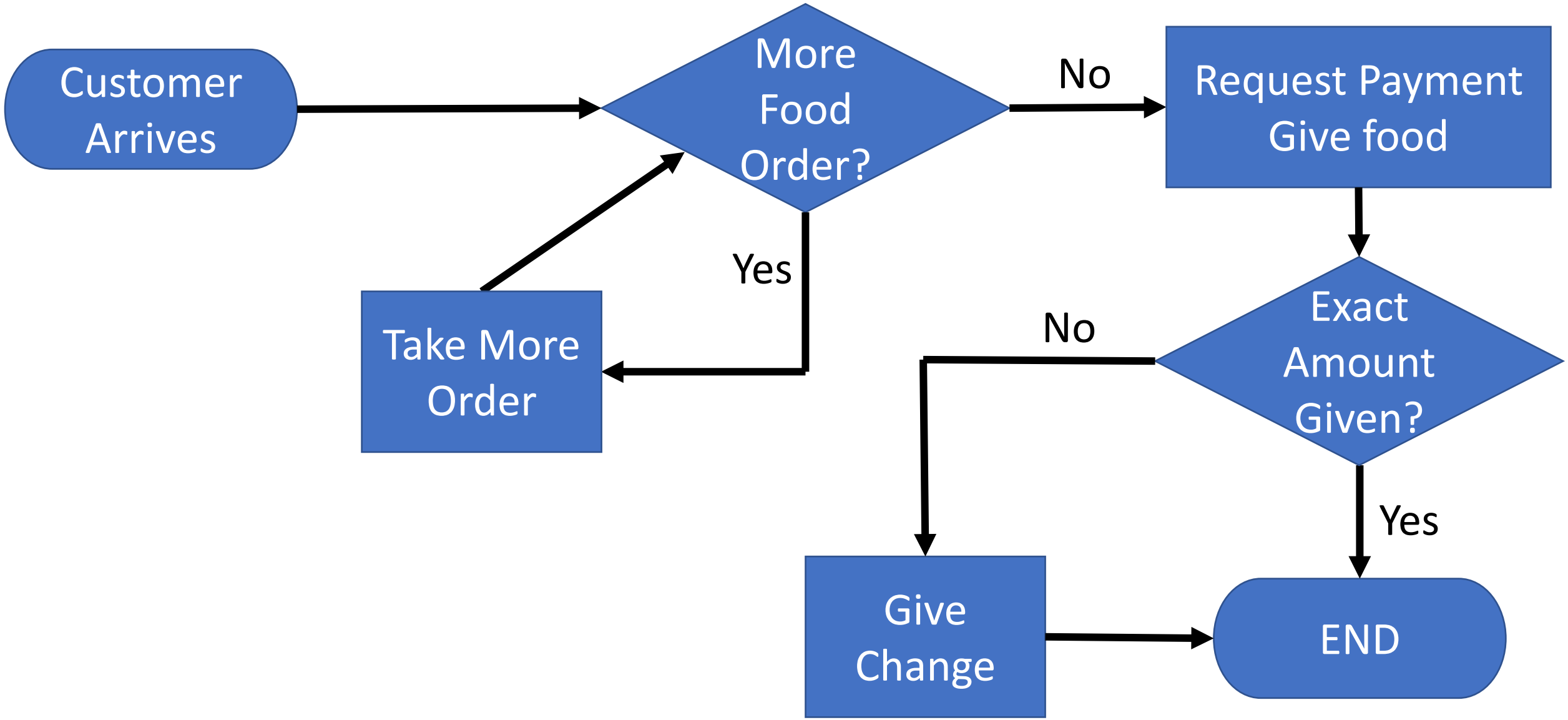
IT1007 TUTORIAL/LAB SESSION 1

Today

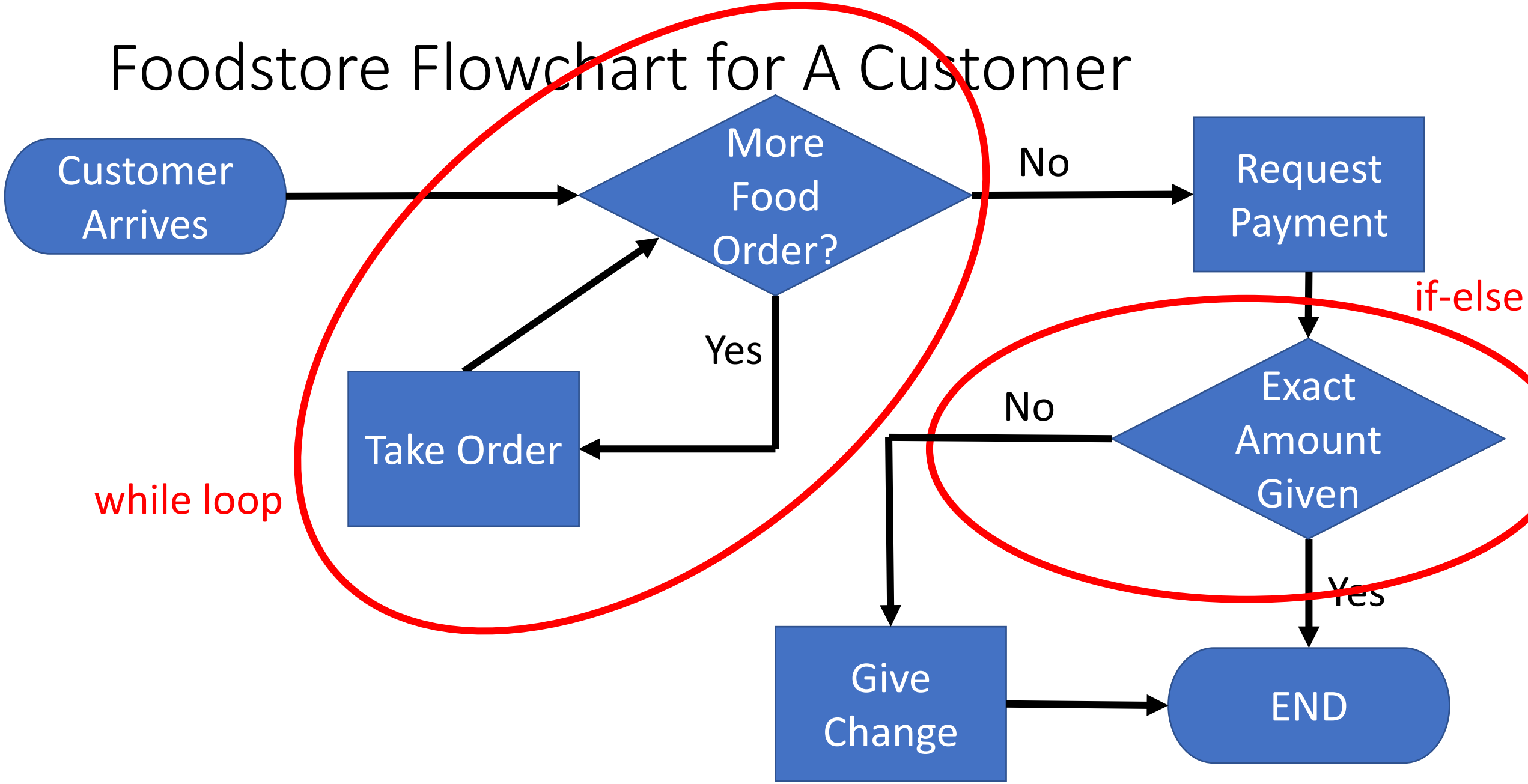
- Flow charts
- String Slicing
- If-else
- Lab Exercise
  - Summation
  - Odd or even

Flowcharts

Foodstore Flowchart for A Customer



Foodstore Flowchart for A Customer



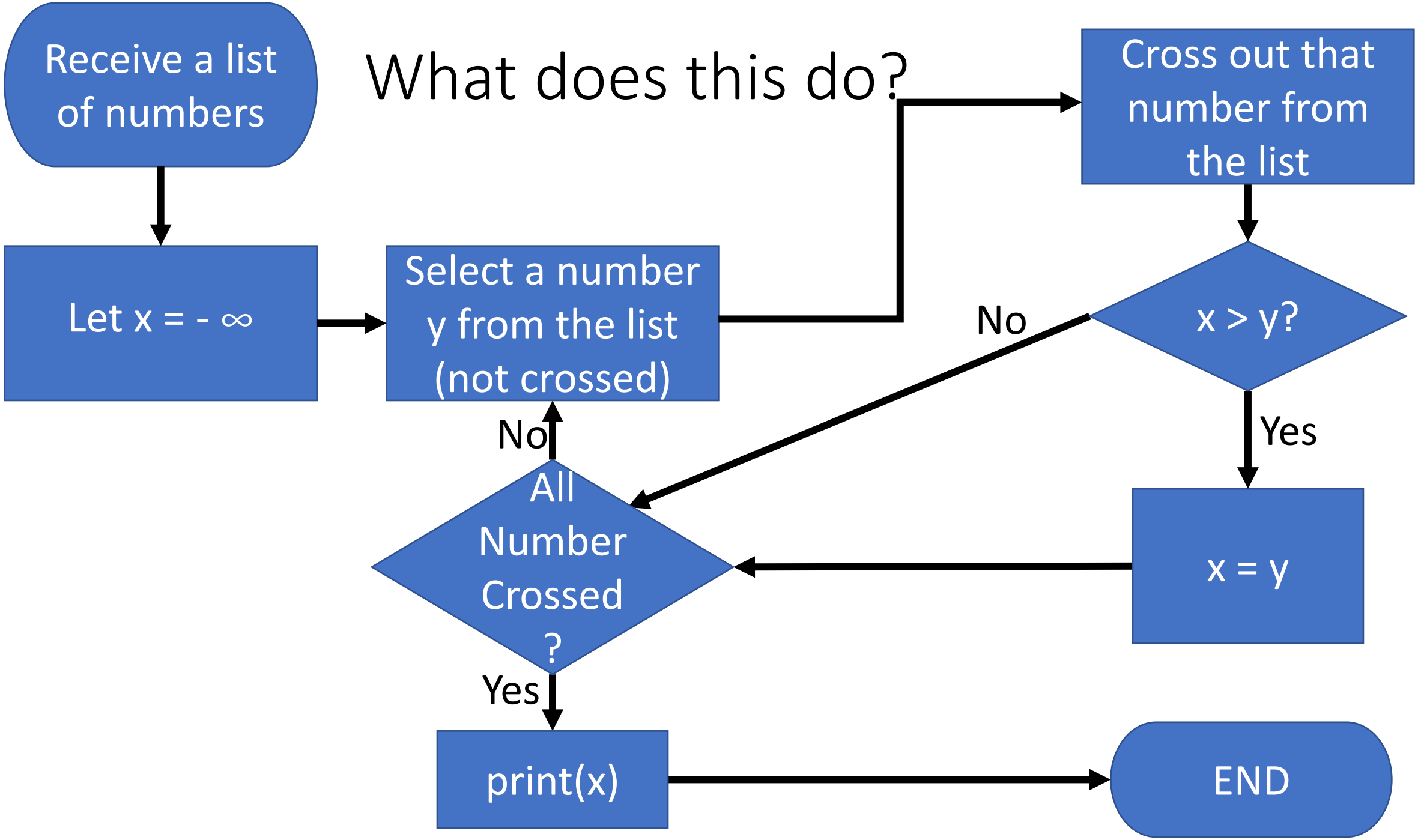
Your turn!

- Create a flow chart to classify animals into different classes
  - For simplicity, we shall assume that there are only 4 types of animals
  - Mammals, Birds, Fishes and Insects

What have you noticed?

Same problem, different solutions

What does this do?



String Slicing

String Slicing

- 0 based indexing (counting starts from 0)
- What is the result if I do the following

```
>>> s = ‘123’
>>> s[0]
‘1’
>>> s[1]
‘2’
>>> s[3]
IndexError: string index out of range
```

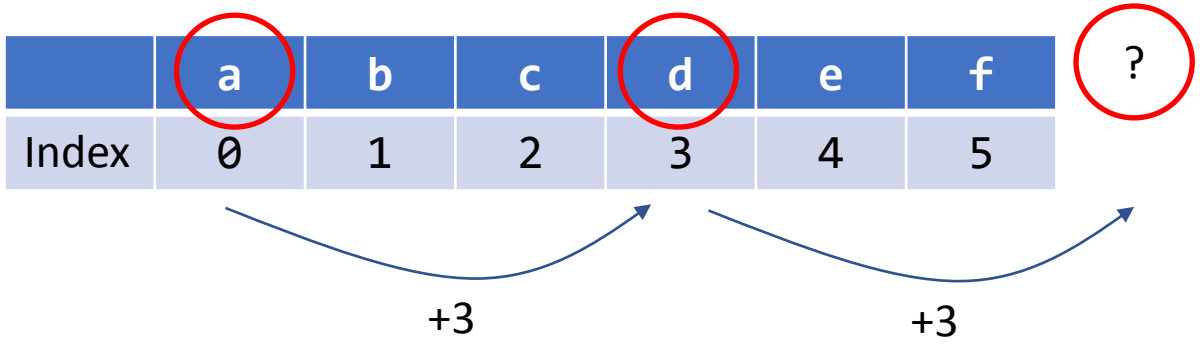
String Slicing [start:stop:step]

- Recall the command for slicing `[start:stop:step]`
- Start – where to start the slice. **Inclusive**.
- Stop – where to stop the slice. **Exclusive.**
- Step – how many steps to “jump”.

- What happens if a parameter is not specified?
  - Start – By default, start from index 0.
  - Stop – By default, include the last letter.
  - Step – By default, “jump” by 1 step.

String Slicing

- Let s = ‘abcdef’
- What is the result of `s[::3]`?
  - ‘ad’



- “Jump” interval is 3. ‘**a**bc**d**ef’

- What is the result of `s[0:6:-1]` and `s[0:4:-1]`?
  - ‘’

Start – By default, start from index 0.
Stop – By default, include the last letter.
Step – By default, “jump” by 1 step.

String Slicing

	a	b	c	d	e	f
Index	0	1	2	3	4	5

- Let s = ‘abcdef’
- What is the result of `s[2]` and `s[2:]` and `s[2::]`? Are they the same?

- Only `s[2:]` and `s[2::]` are the same.

- What happens if we do `s[1:1]`?
  - Get ‘’ (a blank string)

Start – By default, start from index 0.
Stop – By default, include the last letter.
Step – By default, “jump” by 1 step.

String Slicing

	a	b	c	d	e	f
Index	0	1	2	3	4	5

- Let s = ‘abcdef’
- What is the result of `s[]` and `s[:2]` and `s[:2:]`?
- Are they the same?

- Only `s[:2]` and `s[:2:]` are the same.
- `s[]` is a syntax error

Start – By default, start from index 0.
Stop – By default, include the last letter.
Step – By default, “jump” by 1 step.

String Slicing

- Let s = ‘abcdef’
- What about `s[5:0:-1]`?

‘`fedcb`’

- What happens if we do `s[:2:-1]`?

‘`fed`’

- Lecture example: `s[::-1]`

'`fedcba`'

	a	b	c	d	e	f
Index	0	1	2	3	4	5

Start – By default, start from index 0.
Stop – By default, include the last letter.
Step – By default, “jump” by 1 step.

String Slicing

- Let s = ‘abcdef’

	a	b	c	d	e	f
Index	0	1	2	3	4	5

- What happened? By python convention, if step is **negative** default start is the last letter and default stop is the first letter, inclusive.

- Lecture example: s[::-1] is interpreted as “reversing the string”
- 'fedcba’

- Does this mean s[::-1] is the same as s[5:-1:-1]?
- No. s[5:-1:-1] will return a blank string.

Default

- If step > 0
  - Start – By default, start from index 0.
  - Stop – By default, include the last letter.
  - Step – By default, “jump” by 1 step.
- Else (step < 0)
  - Default start = last letter
  - Default end = -1

- Let n = length of your string
- If step > 0
  - Start = 0
  - Stop = n
- Else if step < 0
  - Start = n
  - Stop = -1

if-else

What will it return?

```
def foo():
    if True:
        if False:
            print(1)
        else:
            print(2)
```

A Computer Science thing:
If you don't want to name a function.
Anyhow name it "foo()"

But DON'T do it in practise

if-else

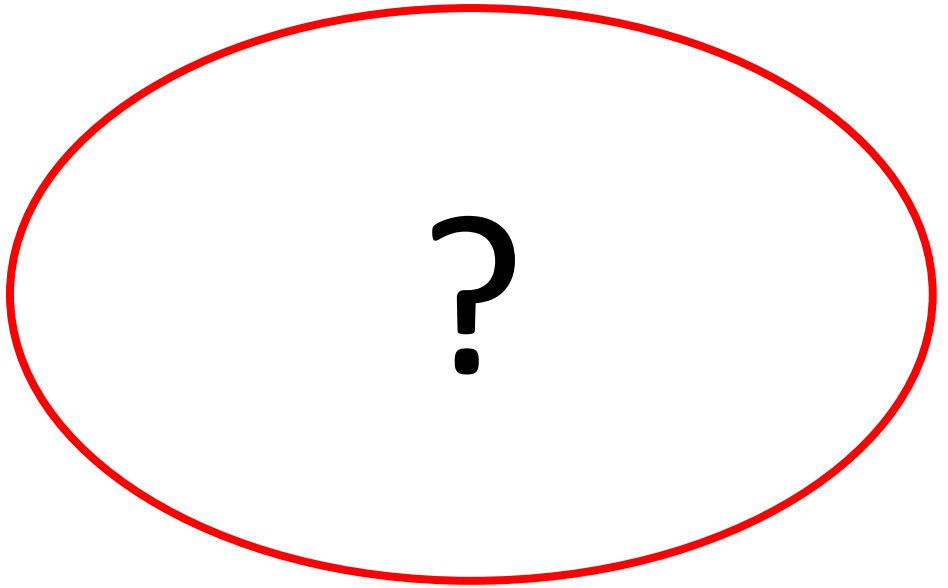
```
def foo():
    if False:
        if True:
            return 1
    else:
        return 2
```

```
def foo():
    if False:
        return 1
    elif False:
        return 2
    elif True:
        return 3
    elif True:
        return 4
    else:
        return 5
```

elif statements will ‘break’ the moment one of them is True

if-else

```
def foo():
    if not True:
        if True:
            print(1)
        else:
            print(2)
```



Be careful with your if-else. You might return nothing!

Can you spot the difference?

Example 1

```
def foo():
     if True:
          if False:
               print(1)
          else:
               print(2)
```

Example 2

```
def foo():
     if True:
          if False:
               print(1)
     else:
          print(2)
```

Comments in Python

Comments in Python

- Usually denoted by # at the start of a line

- Can also be done between pairs of triple quotes

```
#Example of single line comment

'''
Example of triple quotes comment
Wow I can do multiple lines
'''
```

Comments in Python

- Good habit to have comments in your code
- Remind yourself what the code is for
- Help others understand your code

- Remember to make sure you mark out your comments properly. Otherwise, you might get an error when trying to run your program.