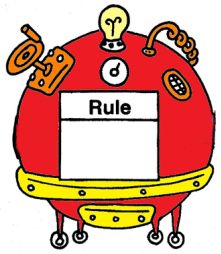
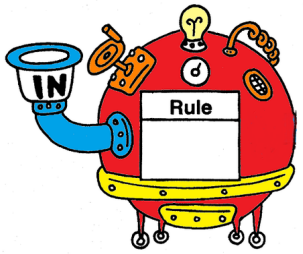
CS-A113 Basics in Programming Y1

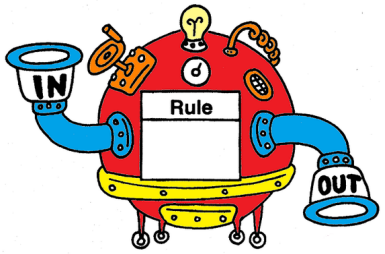
Lecture 27.9.2022



Function Call



Parameters



Return Values

Topics Today

And a tiny bit about software engineering

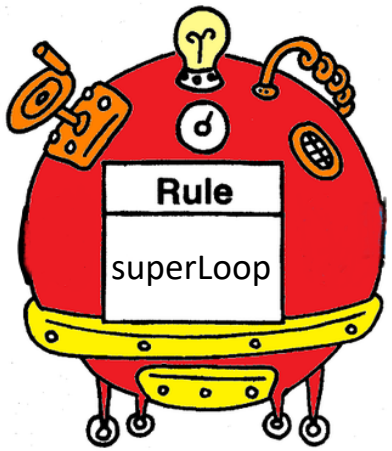
Functions

You already used them:

int(), print(), range()

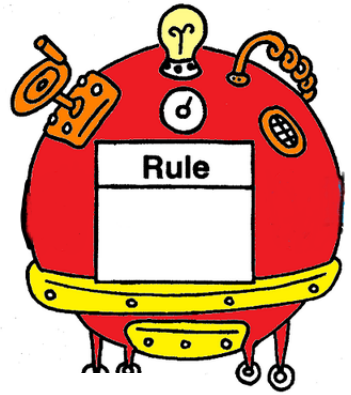
Whatever you defined with **def** became a function

```
def superLoop():
    for i in range(0,50,3):
        print(i)


def main():
    superLoop()
```

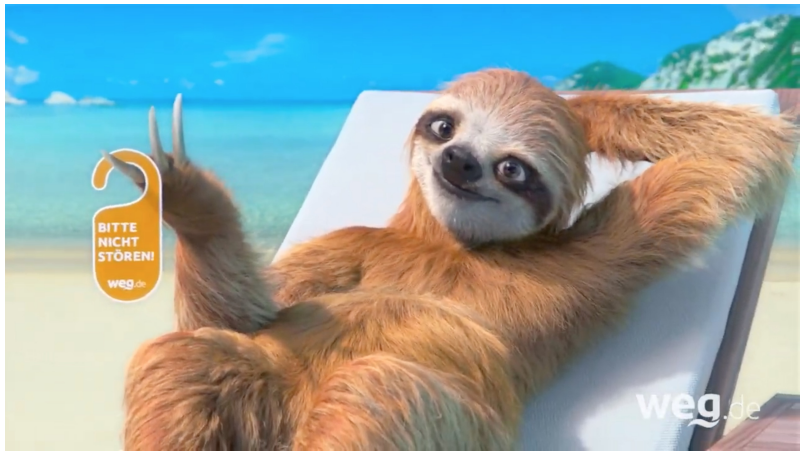


But Why?

Functions: But Why?



Because we are lazy

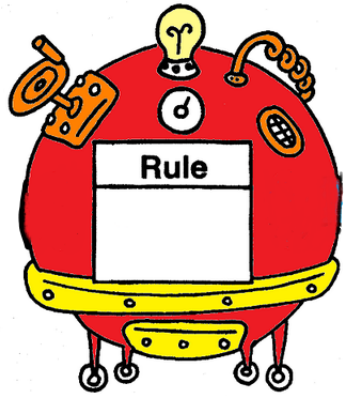


Do you ALWAYS have to reinvent the wheel?

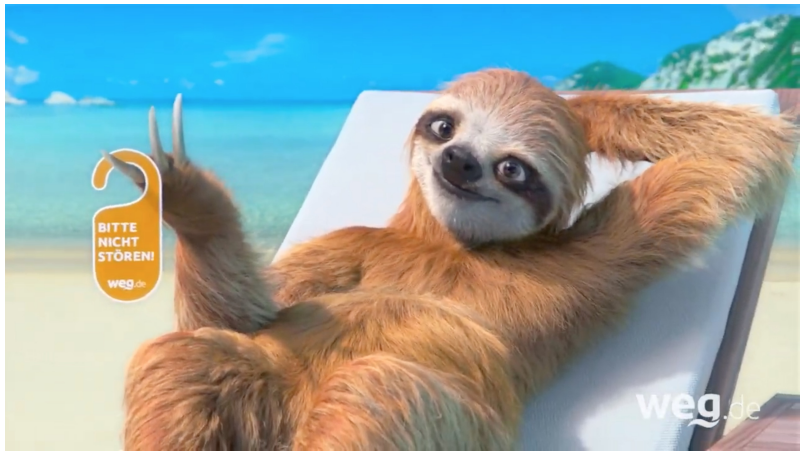
In some cases yes you do, but in most cases it's a big NOOOO!

To structure our Code

Functions: But Why?



Because we are lazy



Do you ALWAYS have to reinvent the wheel?

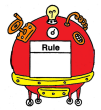
In some cases yes you do, but in most cases it's a big NOOOO!

To structure our Code

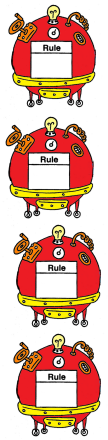
```
def cleanApartment()
   while(notClean):
      stuff=pickUp()
      if stuff== book:
         if bookshelf != full:
            putToBookshelf(stuff)
         else:
            ....
      elif stuff == laundry:
         putToLaundryBasket
         if LaundryBasket == full:
            doLaundry:
      elif stuff == food:
      ...
```

```
def cleanApartment()
   while(notClean):
      stuff = pickUp()
      putAway(stuff)

def putAway(object)
   if object == book:
      ....
   elif object == laundry:
```

Functions

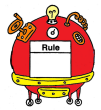


```
def carCosts():
    carGas()
    parkingSpot()
    carInsurance()
    carMaintenance()

    print("Do not forget these costs!")
```

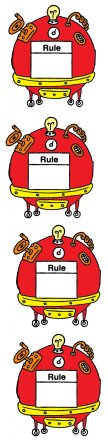


```
def carGas():
    price = 1.39
    print("gas costs",price, "Euros / l")

def parkingSpot():
    price = 120
    print ("parking spot costs",price, "Euros / year")

def carMaintenance():
    lowPrice = 200
    highPrice = 450
    print("maintanence costs ca.",lowPrice,"-",highPrice)

def carInsurance():
    lowPrice = 200
    highPrice = 450
    print("insurance costs ca.",lowPrice,"-",highPrice)
```

Functions

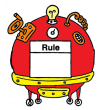


```
def carCosts():
    carGas()
    parkingSpot()
    #carInsurance()
    #carMaintenance()

    print("Do not forget these costs!")
```

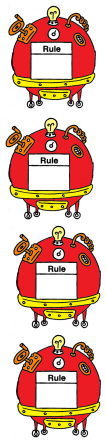


```
def carGas():
    price = 1.39
    print("gas costs",price, "Euros / l")

def parkingSpot():
    price = 120
    print ("parking spot costs",price, "Euros / year")

def carMaintenance():
    lowPrice = 200
    highPrice = 450
    print("maintanence costs ca.",lowPrice,"-",highPrice)

def carInsurance():
    lowPrice = 200
    highPrice = 450
    print("insurance costs ca.",lowPrice,"-",highPrice)
```

Functions

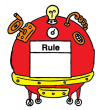


```
def carCosts():
  #carGas()
  #parkingSpot()
   carInsurance()
   carMaintenance()

  print("Do not forget these costs!")
```

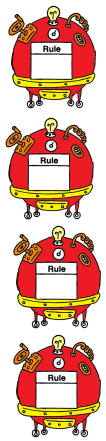


```
def carGas():
    price = 1.39
    print("gas costs",price, "Euros / l")

def parkingSpot():
    price = 120
    print ("parking spot costs",price, "Euros / year")

def carMaintenance():
    lowPrice = 200
    highPrice = 2000
    print("maintanence costs ca.",lowPrice,"-",highPrice)

def carInsurance():
    lowPrice = 200
    highPrice = 450
    print("insurance costs ca.",lowPrice,"-",highPrice)
```

Functions



```
def carCosts():
  #carGas()
  #parkingSpot()
   carInsurance()
   carMaintenance()

  print("Do not forget these costs!")
```



```
def carGas():
    price = 1.39
    print("gas costs",price, "Euros / l")

def parkingSpot():
    price = 120
    print ("parking spot costs",price, "Euros / year")

def carMaintenance():
    lowPrice = 200
    highPrice = 2000
    print("maintanence costs ca.",lowPrice,"-",highPrice)

def carInsurance():
    lowPrice = 200
    highPrice = 450
    print("insurance costs ca.",lowPrice,"-",highPrice)
```

Functions

- A named subprogram

```
def function_name():
    code…
```

```
def another_function():
    code…
```

```
code…
```

Independent blocks of program code

Functions

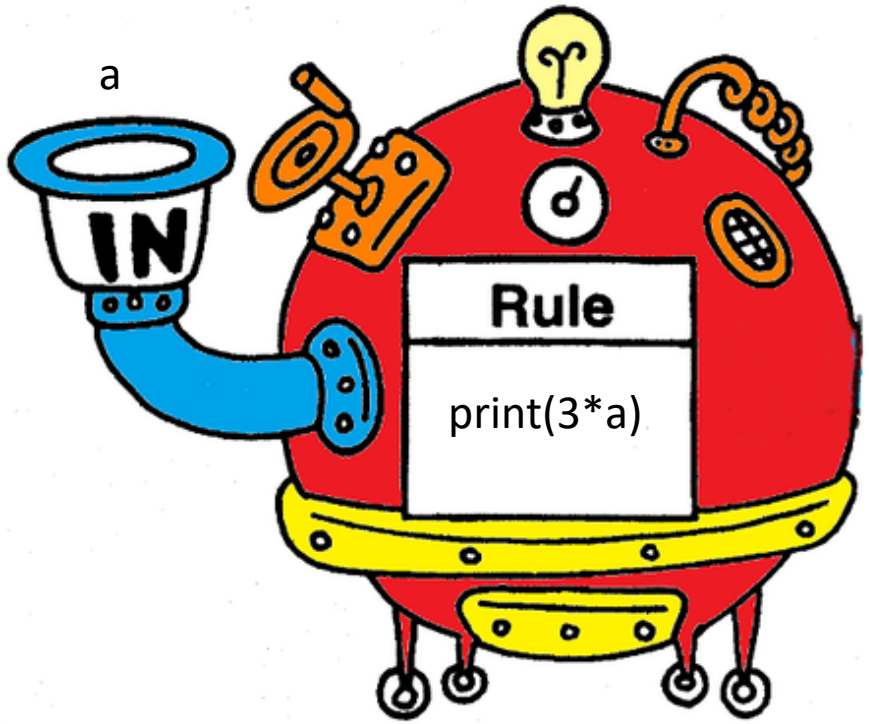
```
def function_name():
    code…

def another_function():
    function_name()
    code…

main()
    function_name()
    another_function()
```

Functions can be ***called*** from elsewhere in the code

Parameters

```
def myFunction(a):
    print(3*a)
```

myFunction

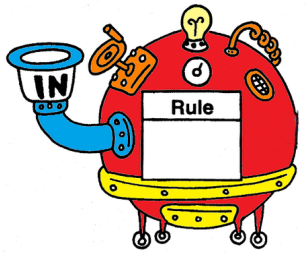



```
def main():
    b = 5
    myFunction(7)
    myFunction(b)
```

21

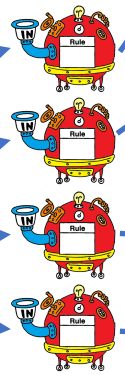
15

Parameters



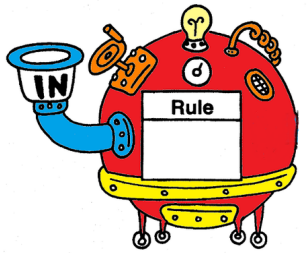
Car example

```
def carCosts():
    distance = 15
    location = ”cityCenter”
    type = ”full”
    carAge = 3
    place = ”to be”

    carGas(distance)
    parkingSpot(location)
    carInsurance(type)
    carMaintenance(carAge)

    print(”Do not forget these costs!”)
```



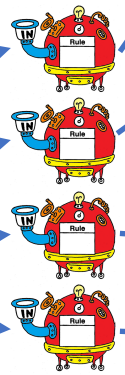
```
def carGas(distance):
    gasPrice = 1.39
    kmPerL = 15
    cost = way/kmPerL*gasPrice
    print(”Gas costs”,cost,”for driving”,way)

def parkingSpot(place):
    if place == ”cityCenter”:
        cost = 450
    elif place == ”forest”:
        cost = 0
    else:
        cost = 200
    print (”your parking spot costs”,cost, ”Euros / year”)

def carInsurance(level):
    ...

def carMaintenance(age)
    ....
```

Parameters



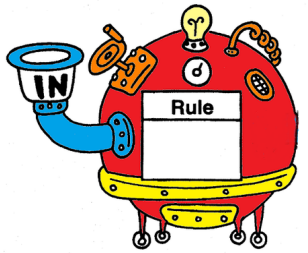
Car example

```
def carCosts():
    distance = 15
    location = ”cityCenter”
    type = ”full”
    carAge = 3
    place = ”to be”

    carGas(distance)
    parkingSpot(location)
    carInsurance(type)
    carMaintenance(carAge)

    print(”Do not forget these costs!”)
```



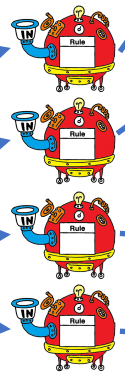
```
def carGas(distance):
    gasPrice = 1.39
    kmPerL = 15
    cost = distance/kmPerL*gasPrice
    print(”Gas costs”,cost,”for driving”,distance)

def parkingSpot(place):
    if place == ”cityCenter”:
        cost = 450
    elif place == ”forest”:
        cost = 0
    else:
        cost = 200
    print (”your parking spot costs”,cost, ”Euros / year”)

def carInsurance(level):
    ...

def carMaintenance(age)
    ....
```

Parameters



Car example

```
def carCosts():
    distance = 15
    location = "cityCenter"
    type = "full"
    carAge = 3
    place = "to be"

    carGas(distance)
    parkingSpot(location)
    #carInsurance(type)
    #carMaintenance(carAge)

    #print("Do not forget these costs!")
```



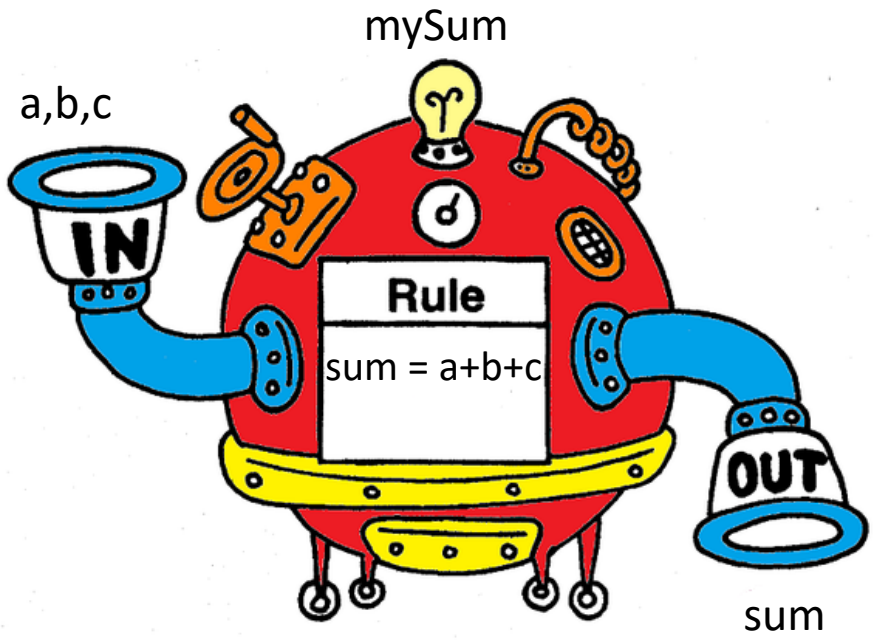
```
def carGas(distance):
    gasPrice = 1.39
    kmPerL = 15
    cost = distance/kmPerL*gasPrice
    print("Gas costs",cost,"for driving",distance)

def parkingSpot(place):
    if place == "cityCenter":
        cost = 450
    elif place == "forest":
        cost = 0
    else:
        cost = 200
    print ("your parking spot costs",cost, "Euros / year")

def carInsurance(level):
    ...

def carMaintenance(age)
    ....
```

Parameters

- Input for the function
- Can be none, one or more

```
def my_function(a, b, c):
    print(a, b, c)
```

Return Values

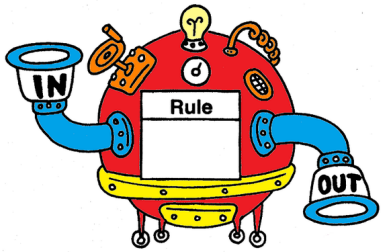



```
def mySum(a,b,c):
    sum = a+b+c
    return sum
```

```
def main():
    startAge = 5
    education1 = 6
    education2 = 6
    birthYear = 1996
    masterDuration = 5
    ageAtUni = mySum(startAge,education1,education2)
    graduationYear = mySum(birthYear,ageAtUni,masterDuration)
    print("I was born",birthYear,"and graduated in", graduationYear)
```

I was born 1996 and graduated in 2018

Return Value



Car example

```
def carCosts():
    distance = 15
    location = ”cityCenter”
    type = ”full”
    carAge = 3
    place = ”forest”

    cost=0
    cost += carGas(distance)
    cost += parkingSpot(location)
    cost += carInsurance(type)
    cost += carMaintenance(carAge)

    print(”Your car costs:”, cost)
```

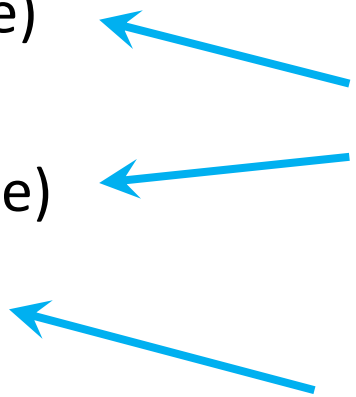
```
def carGas(distance):
    gasPrice = 1.39
    kmPerL = 15
    cost = distance/kmPerL*gasPrice
    return(cost)

def parkingSpot(place):
    if place == ”cityCenter”:
        cost = 450
    elif place == ”forest”:
        cost = 0
    else:
        cost = 200
    return(cost)

def carInsurance(level):
    ...

def carMaintenance(age)
    ....
```

Return Value

- Function can return a result
  - Usually should, too, at least a status code
- The value can be any variable
- Command "***return***" ends the function
  - return(5)

Return Value

```
def is_equal(a, b):
    if a == b:
        return(True)
    else:
        return(False)
    print(a, b)
```

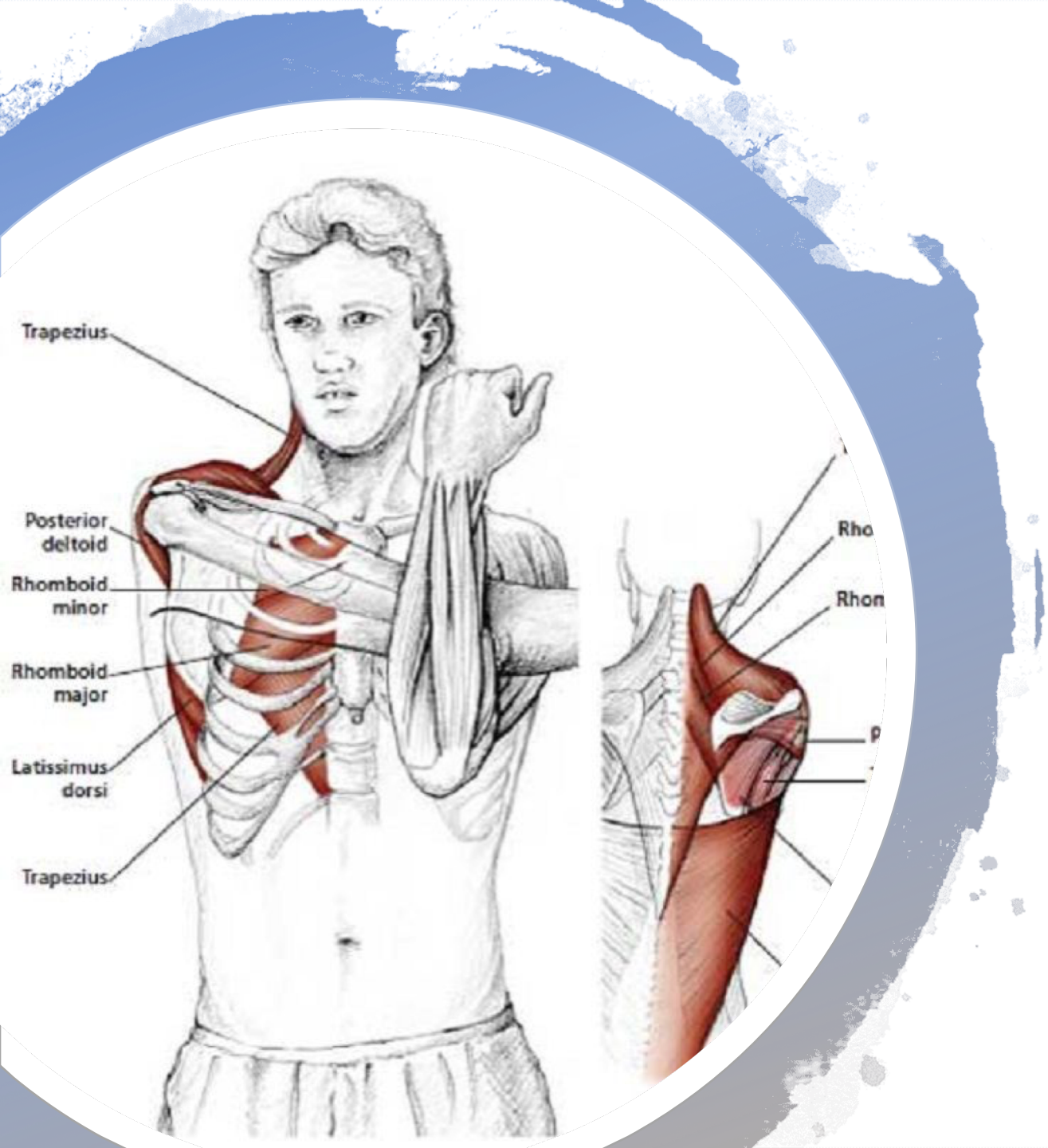


Two return statements, of type boolean

print is never reached, bad code

Think:

Is this function needed at all?



Break: Move your Shoulders

Coding Examples

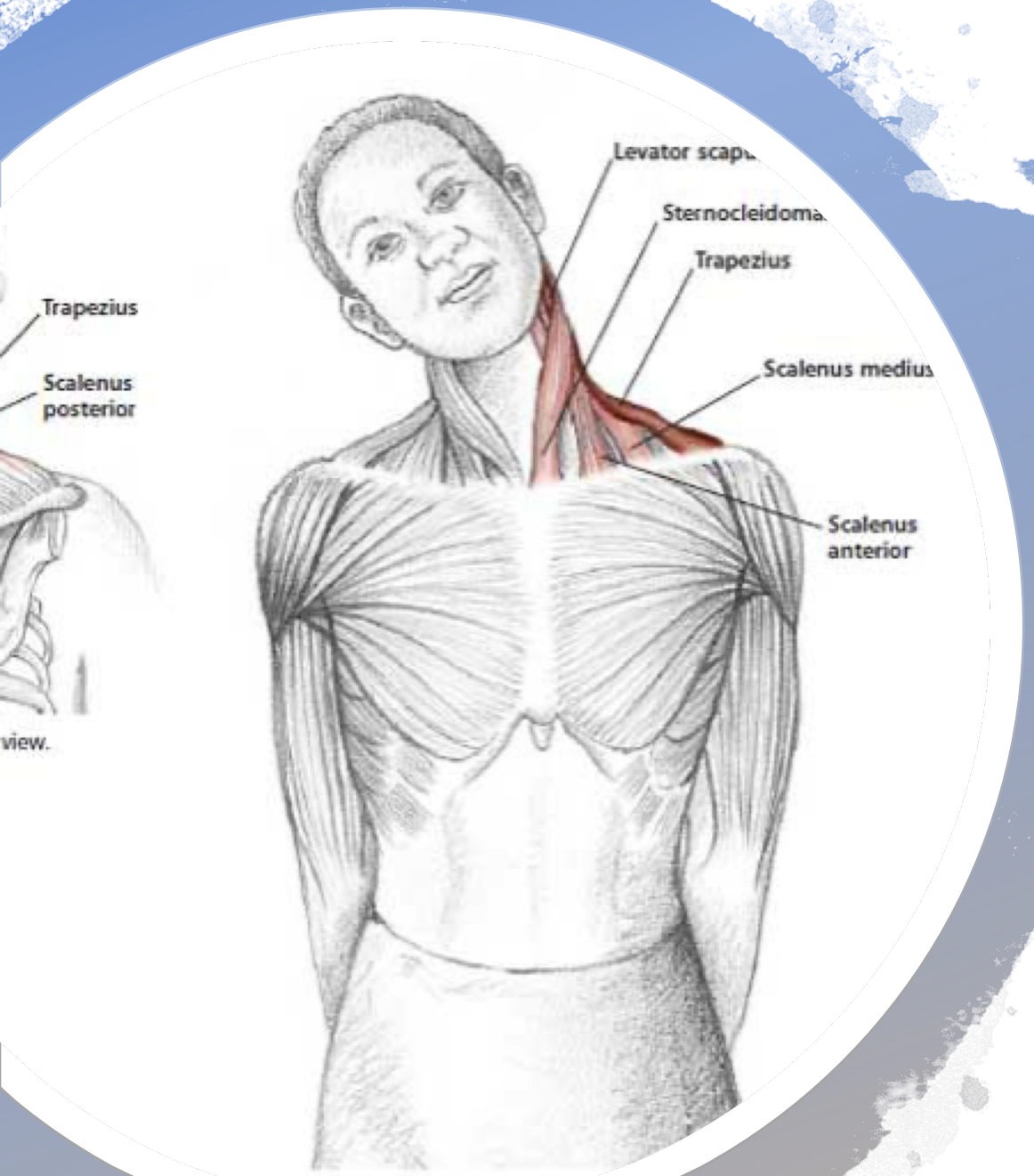
```
def carCosts():
    distance = 15
    location = ”cityCenter”

    cost =0
    cost += carGas(distance)
    cost += parkingSpot(location)

print(”Your car costs:”, cost)
```

```
def carGas(way):
    gasPrice = 1.39
    kmPerL = 15
    cost = distance/kmPerL*gasPrice
    return(cost)


def parkingSpot(place):

    if place == ”cityCenter”:
        price = 450
    elif place == ”forest”:
        price = 0
    else:
        price = 200
    return(price)
```

Coding Examples

```
def carCosts():
    distance = 15
    location = ”cityCenter”
    type = ”full”
    carAge = 3
    place = ”forest”

    cost =0
    cost += carGas(30)
    cost += parkingSpot(location)
    cost += carMaintenance(carAge)

print(”Your car costs:”, cost)
```

```
def carGas(distance):
    gasPrice = 1.39
    kmPerL = 15
    cost = distance/kmPerL*gasPrice
    return(cost)


def parkingSpot(place):
    if place == ”cityCenter”:
        price = 450
    elif place == ”forest”:
        price = 0
    else:
        price = 200
    return(price)


def carMaintenance(age):
    cost = age*100

    return(cost)
```



Break:
Move your Neck!

Function

- A named subprogram
- Input: parameters
- Code processes input
- Output: returned value

Why Functions?

- Reusability
  - Same code needed in many places
  - E.g., validating social security number
- Modularity
  - Organize a program to separate sections
- Reliability
  - Re-using well tested and well defined functions avoids problems

How to Name?

- Use descriptive names
  - car_cost_gas(), car_cost_parking()
- Follow a style
  - Shared projects usually have a style guide
  - With Python, lowercase names with underscores common
- Using good names makes programs easier to understand and maintain
  - Not: my_function_1(), my_functio_1(), my_function2()…

Modular Architecture

- Larger programs have lots of code
  - 10 000 lines not unusual, 1 000 000 not unheard
- Sub-parts of the program are called ***modules***
  - Usually kept in separate files and maintained by separate programmers
- Modules communicate through ***interfaces***
  - E.g. function calls
- A module often has some functions that should be called and others that are internal
  - This is the start of ***architecture***

What Goes Where?

- How to know what each function should do?
- Analysis, understanding the problem the program solves
  - Stop writing code and start to think
- Look for commonalities, related tasks, independent tasks
- Skill grows with experience
- Pen and paper or a whiteboard are common tools
- Also formal modeling tools, like Unified Modeling Language

My Code is a Mess

- Working without a plan
- Or requirements change, plan was not complete
- Time to ***refactor***
  - Re-arrange the code and functions differently
- Refactoring is (or should be) common in ***agile*** projects
  - A ***sprint*** is used to clean up the code
  - No functionality is added
  - reduces ***maintenance debt***


"That's all Folks!"
Isberg