Smith College

Computer Science

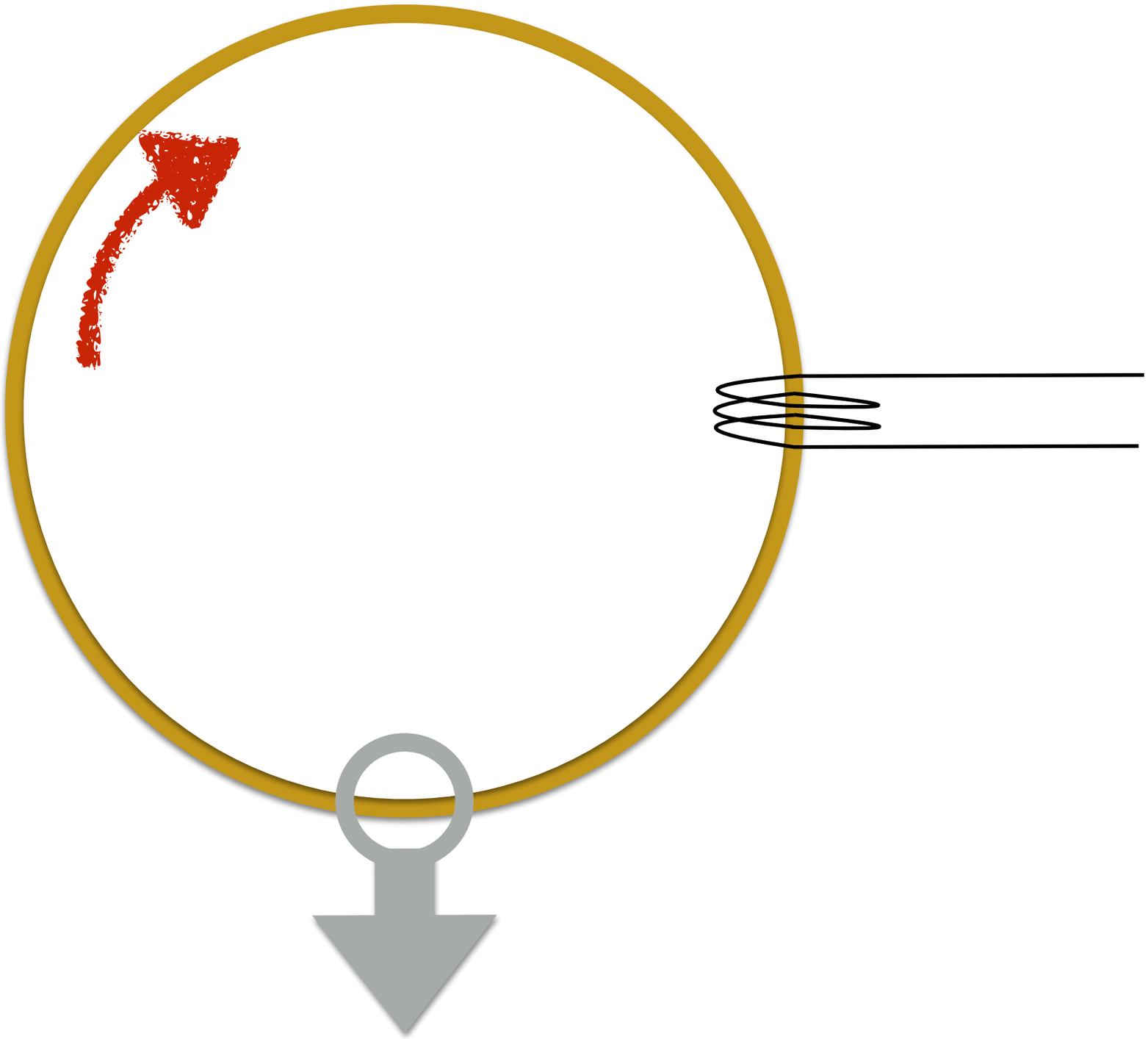
Lecture Notes
CSC111

Week 5 — Spring 2018

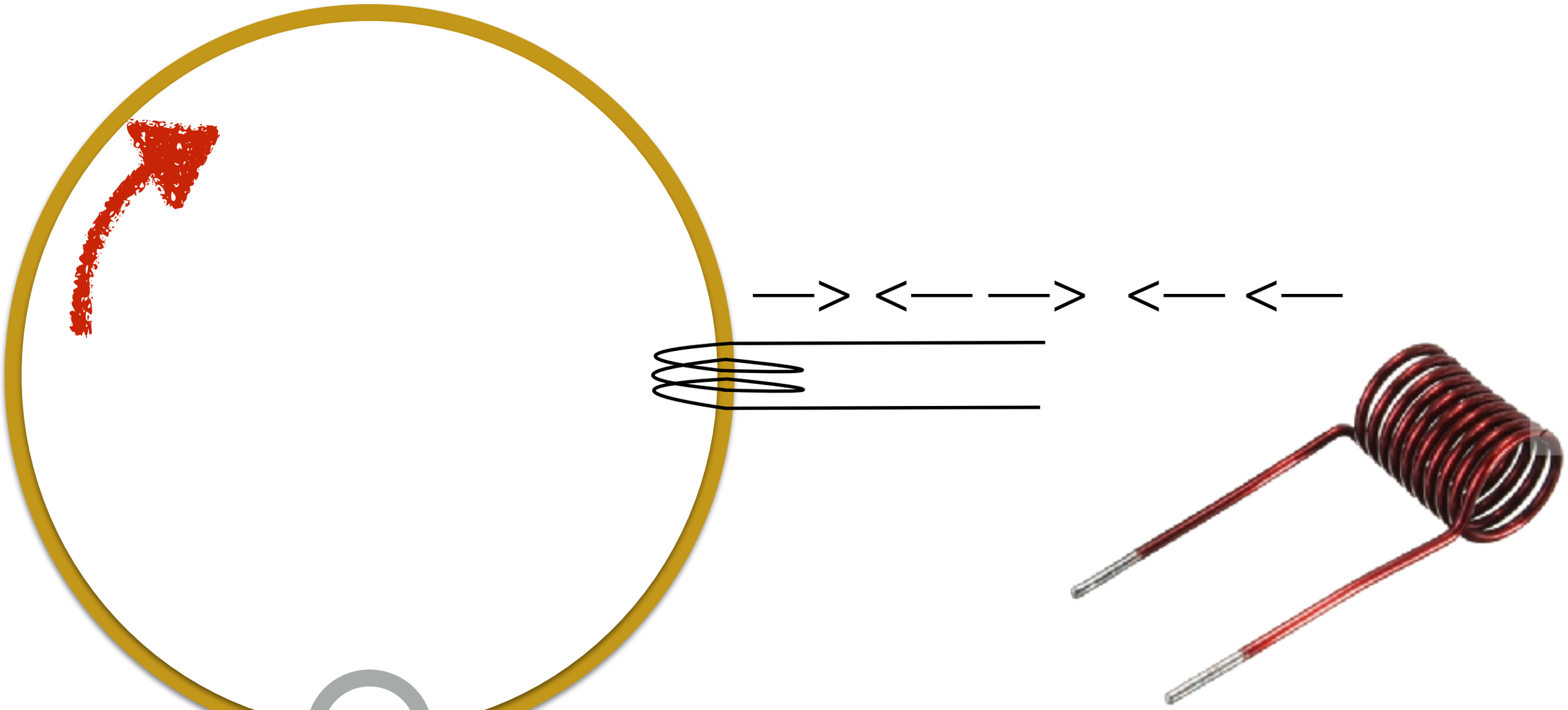
Dominique Thiébaut
dthiebaut@smith.edu

File Processing

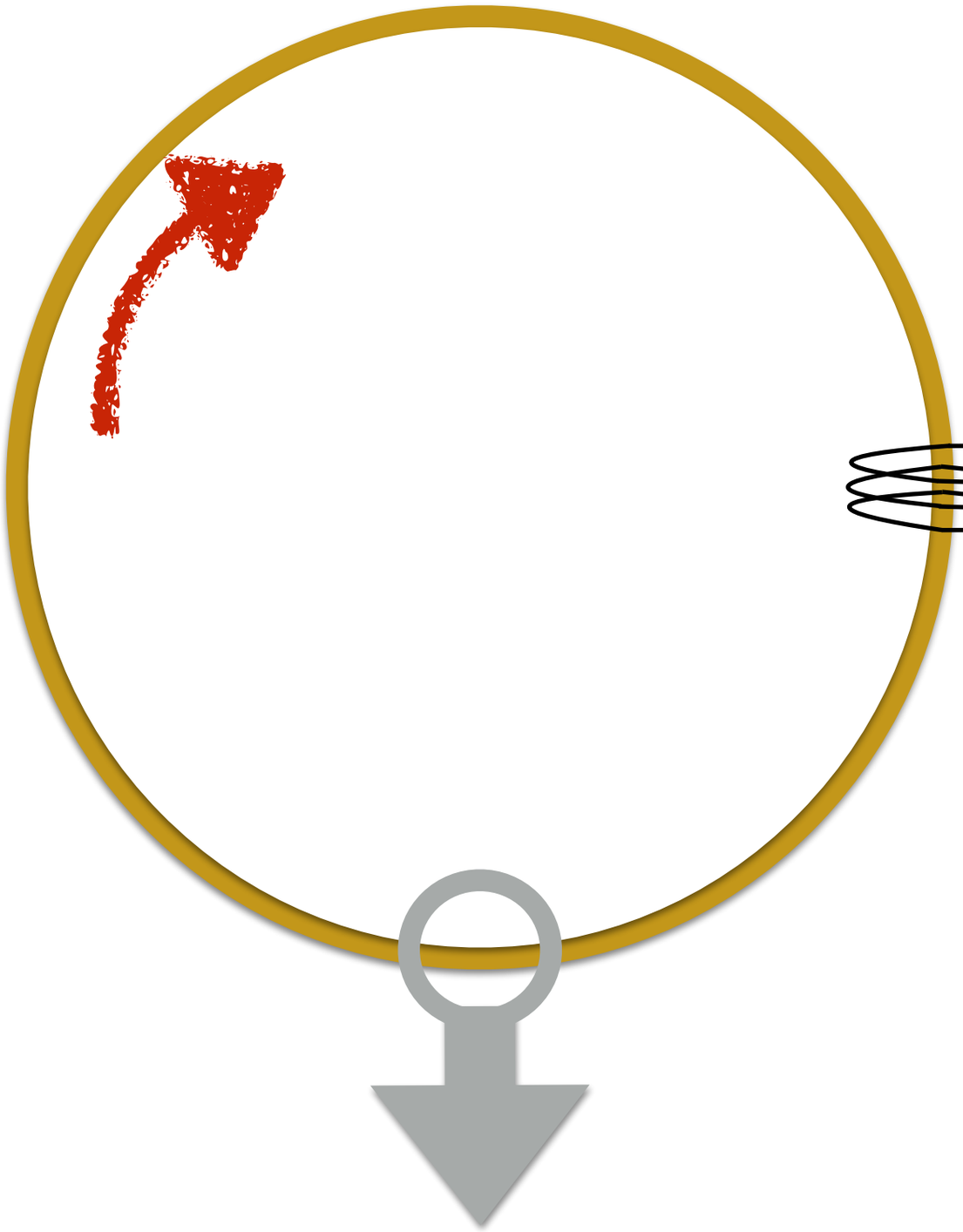
How a Hard Disk Works

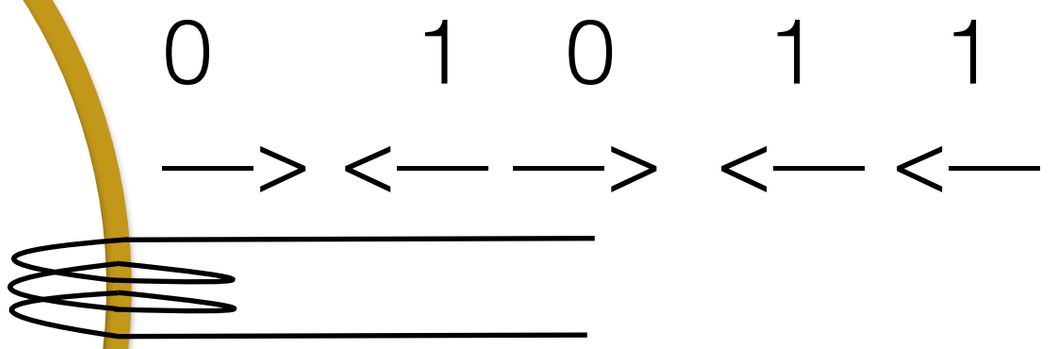


N	S	N	N	N	S	N	S	N	S	N	N	N	N	S	N
S	N	S	S	S	N	S	N	S	N	S	S	S	S	N	S

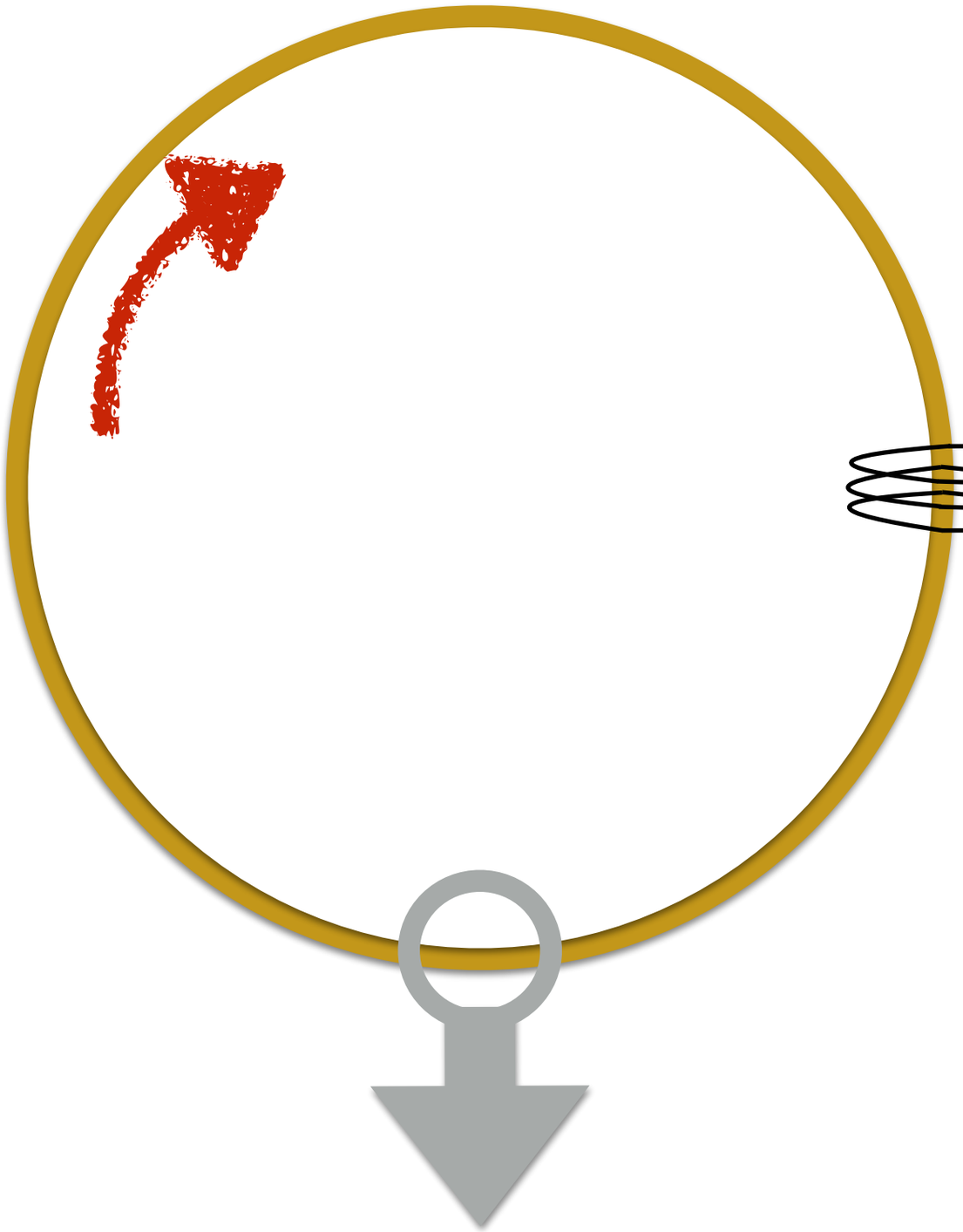


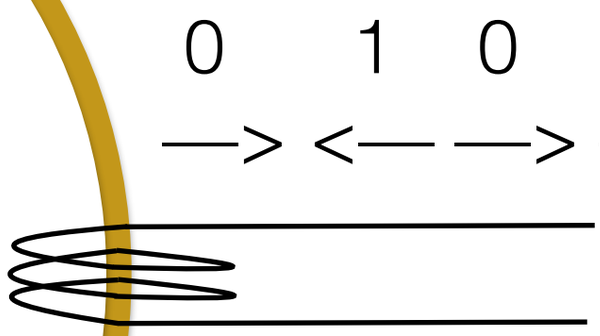
N	S	N	N	N	S	N	S	N	S	N	N	N	N	S	N
S	N	S	S	S	N	S	N	S	N	S	S	S	S	N	S



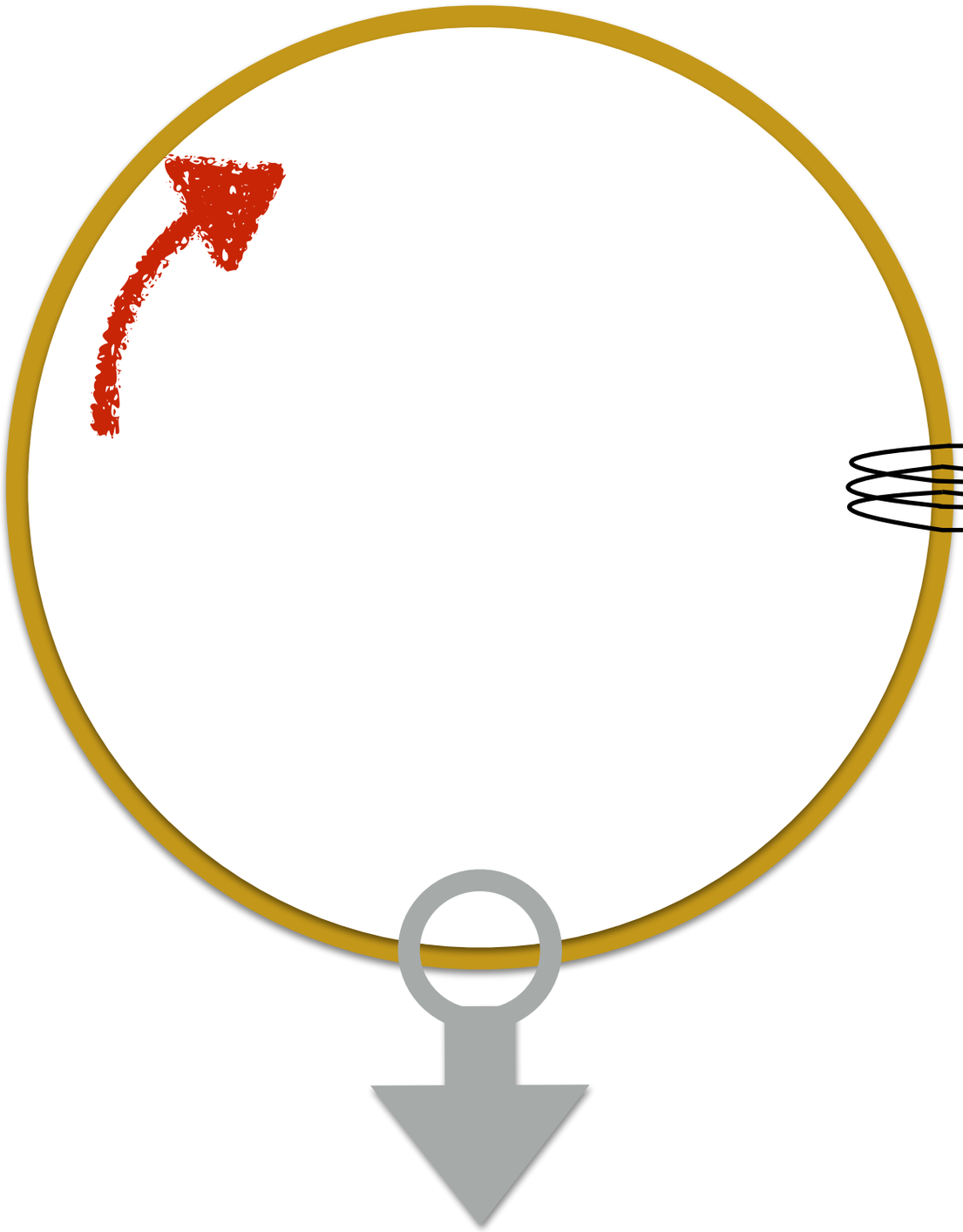


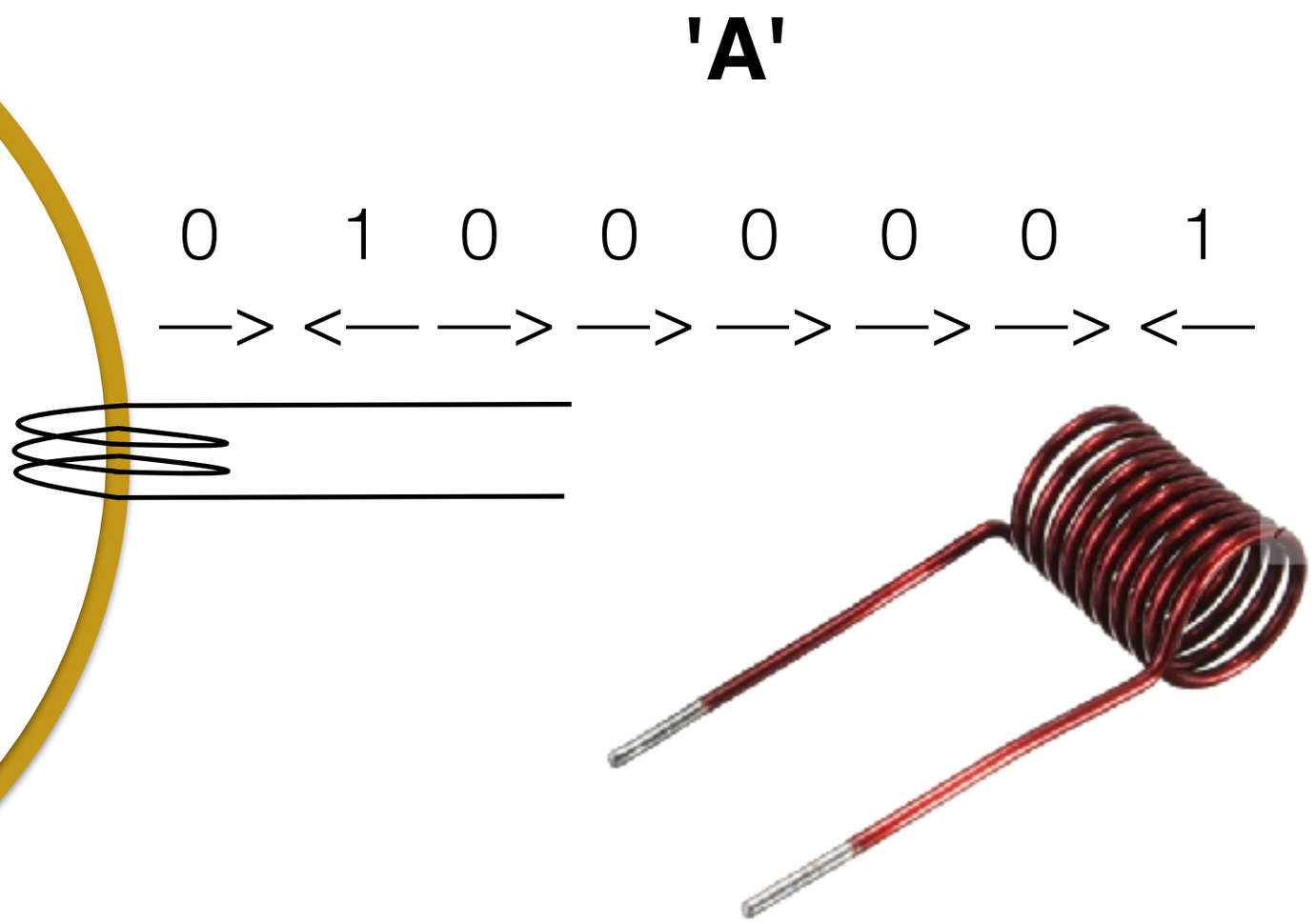
N S	S N	N S	N S	N S	S N	N S	S N	N S	S N	N S	N S	N S	N S	S N	N S





N	S	N	N	N	S	N	S	N	S	N	N	N	N	S	N
S	N	S	S	S	N	S	N	S	N	S	S	S	S	N	S





N S	S N	N S	N S	N S	S N	N S	S N	N S	S N	N S	N S	N S	N S	S N	N S


RA60p

What are Files?

- Containers of bits, organized in bytes

- May contain text, images, music, movies, programs, applications, list of files (folders)…

- In Python, we will first play with **text files**

Mini Lab

- Create a file containing following text (use Notepad or TextEdit):

  Strength is the capacity to break
  a Hershey bar into four pieces
  with your bare hands - and then
  eat just one of the pieces.
  + Judith Viorst

- Save it under the name ***chocolate.txt*** in the same directory where you store your python programs

- Write the following Python program:

*readChocolateFile.py - /Users/thiebaut/Desktop/Dropbox/111/readChocolateFile.py (...

```
# readChocolateFile.py
# D. Thiebaut
# Opens a text file and displays it contents.

def main():
    # open file
    file = open( "chocolate.txt", "r" )

    # read each line from file and display it
    for line in file:
        print( line )

    # close the file
    file.close()

main()
```

Ln: 13 Col: 20

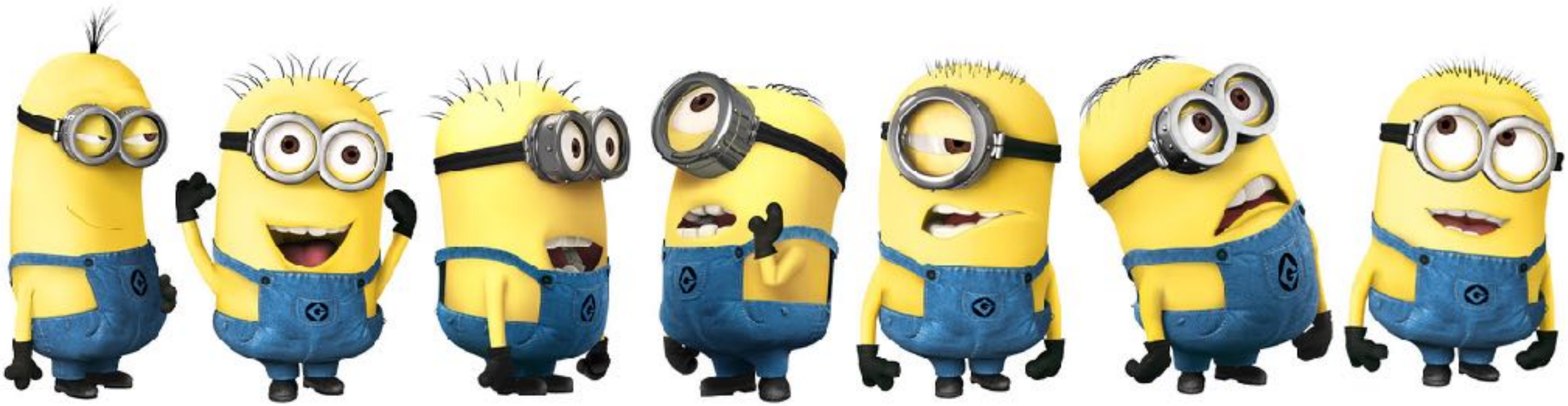
Chapter 6

Defining Functions

Outline

- **Functions**
- **Function Parameters**
- **Functions Returning Values**

Functions



are your minions!

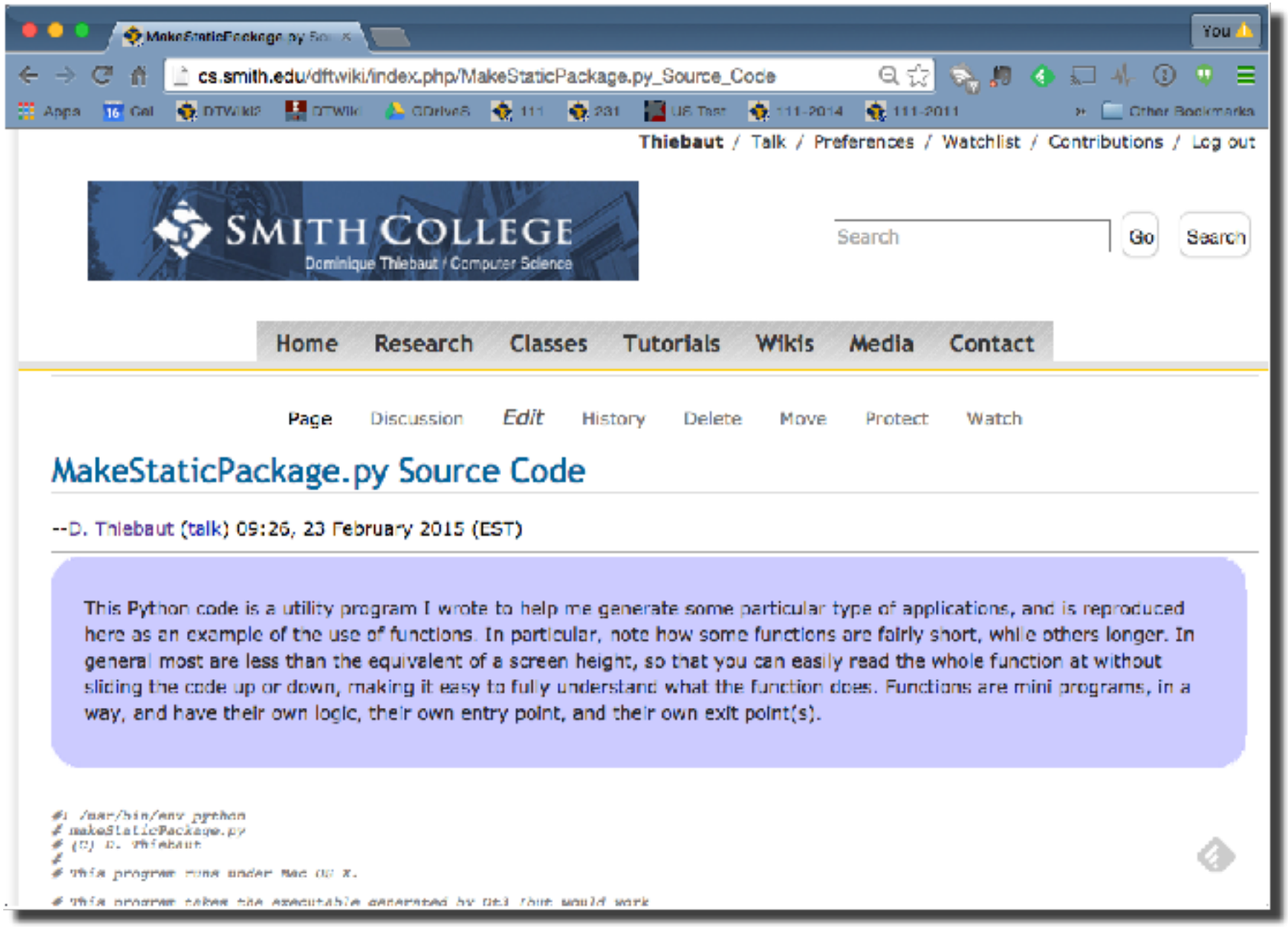
They have names…
(def main)

They work only if you call them…
main()

They can work on different things…

```
daveWorkOn( "Banana" )
daveWorkOn( "Lots of Bananas" )
```

A Real Example

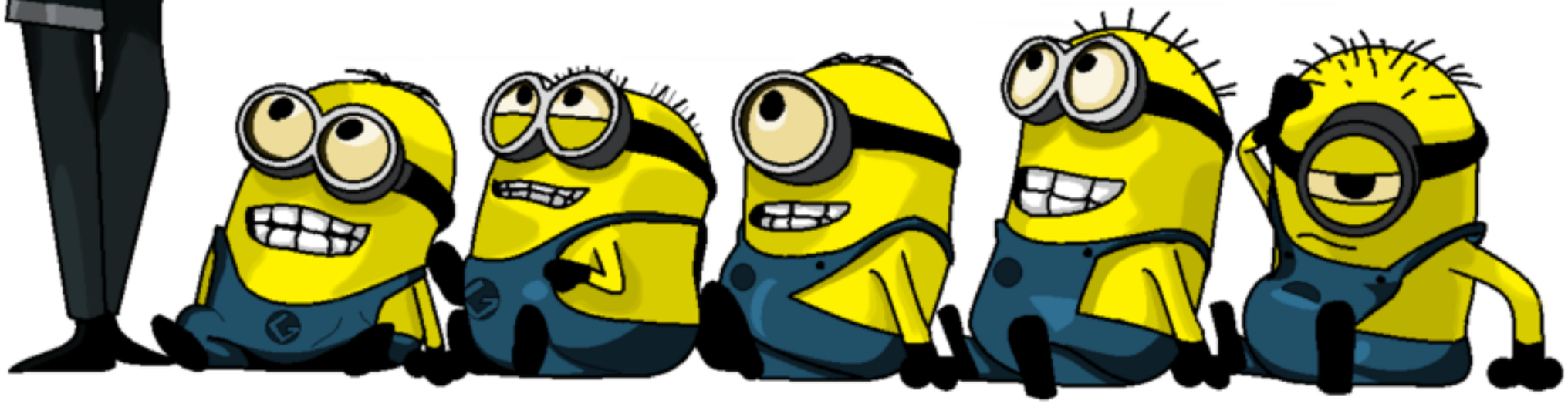


http://cs.smith.edu/dftwiki/index.php/MakeStaticPackage.py_Source_Code

main()

functions

dave() carl() stuart()



Two different types of functions:

Some functions always do the same thing

```
def printBar():
    print( 60 * '-' )


def sayHello():
    print( )
    print( "Hello, and welcome!" )
    print( )
```

Some functions adjust their behavior depending on what we give them to work with.

Dave

work for Dave to perform:

DaveEat(*fruit*)

- open mouth
- put *fruit* in mouth
- chew
- swallow

work for Dave to perform:

DaveEat(*fruit*)

- open mouth
- put *fruit* in mouth
- chew
- swallow

DaveEat(banana)

work for Dave to perform:

DaveEat(*fruit*)

- open mouth
- put *fruit* in mouth
- chew
- swallow

DaveEat(banana)

DaveEat(orange)

work for Dave to perform:

DaveEat(*fruit*)

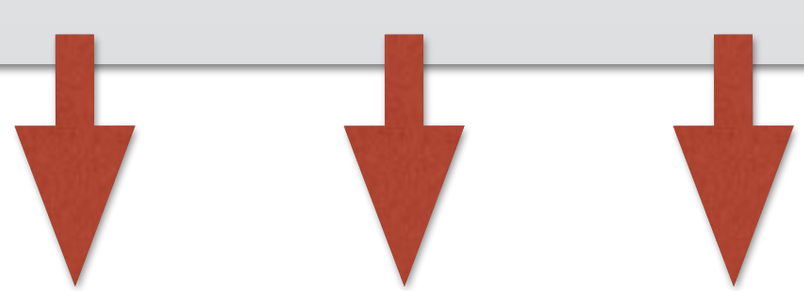
- open mouth
- put *fruit* in mouth
- chew
- swallow

DaveEat(banana)

DaveEat(orange)

DaveEat(apple)

```
def DaveEats( fruit ):
    print( "Dave opens his mouth" )
    print( "and eats the", fruit )



def main():
    daveEats( "banana" )

main()
```



Dave opens his mouth
and eats the banana

```
def daveEats( fruit ):
    print( "Dave opens his mouth" )
    print( "and eats the", fruit )



def main():
    daveEats( "banana" )
    daveEats( "apple" )
main()
```

Dave opens his mouth
and eats the banana
Dave opens his mouth
and eats the apple

passing of parameter

```
def daveEats( fruit ):
    print( "Dave opens his mouth" )
    print( "and eats the", fruit )


def main():
    daveEats( "banana" )
    daveEats( "apple" )

main()
```

Formal Parameter

variable

passing of parameter

```
def daveEats( fruit ):
    print( "Dave opens his mouth" )
    print( "and eats the", fruit )



def main():
    daveEats( "banana" )
    daveEats( "apple" )

main()
```

Actual Parameter

fruit = "banana"

```
def daveEats( fruit ):
    print( "Dave opens his mouth" )
    print( "and eats the", fruit )


def main():
    daveEats( "banana" )
    daveEats( "apple" )

main()
```

Another Example

```
def printBar( char, length ):
    print( char * length )

def main():
    printBar( "#", 10 )


main()
```

Another Example

```
def printBar( char, length ):
    print( char * length )

def main():
    printBar( "#", 10 )


main()
```

```
##########
```

Another Example

```
def printBar( char, length ):
    print( char * length )

def main():
    printBar( "#", 10 )
    printBar( "a", 5 )

main()
```

```
##########
```

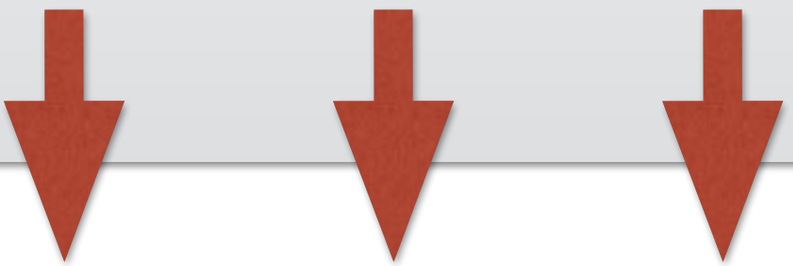
Another Example

```
def printBar( char, length ):
    print( char * length )

def main():
    printBar( "#", 10 )
    printBar( "a", 5 )

main()
```

```
##########
aaaaa
```

Challenge

```
def printBar( char, length ):
    print( char * length )

def main():
    printBar( "#", 10 )
    printBar( "a", 5 )
    printBar( 3, 5 )
main()
```



Write a program with functions that sings happy birthday to **Dave.**

Write a program that sings happy birthday to **Dave** and to **Stuart.**

Modify the program so that it sings happy birthday to **Dave**, **Stuart**, **Jerry**, **Jorge**, **Tim**, **Mark**, **Phil**, **Kevin**, and **Jon.**

Fun modification:
Modify the program so that it sings happy birthday, and formats the song with bars made with the person's name, e.g. `*Dave*Dave*Dave*Da…` above and below the song.

We stopped here last time…

Review

Line-Up your Minions!

Acting Out a Program

Functions Returning Values

Exercises

A Word on Testing

A Different Look at Passing Parameters and Returning Values

Review

Line-Up your Minions!

Acting Out a Program

Functions Returning Values

Exercises

A Word on Testing

A Different Look at Passing Parameters and Returning Values

Line up your Minions!!!

Line up your Minions!!!
Functions

Line up your ~~Minions~~!!!

Functions

```
# header
# header

def someName():
    xxxx
    xxxx

def someOtherName():
    xxxx
    xxxx

def main():
    xxxx
    xxxx

main()
```

Line up your Minions!!!

Functions

```
# header
# header

def someName():
    xxxx
    xxxx someOtherName()
    xxxx

    def someOtherName():
        xxxx
        xxxx

def main():
    xxxx
    xxxx

main()
```

Line up your Minions!!!

Functions

```
# header
# header

def someName():
    xxxx
    xxxx someOtherName()
    xxxx
    def someOtherName():
        xxxx
        xxxx

def main():
    xxxx
    xxxx

main()
```

BAD!!!!

Line up your ~~Minions~~!!!

Functions

```
# header
# header

def someName():
    xxxx
    xxxx

def someOtherName():
    xxxx
    xxxx someName()

def main():
    xxxx
    xxxx

main()
```

Good!!!!!

GOOD

BAAAD!

Review

Line-Up your Minions!

Acting Out a Program

Functions Returning Values

Exercises

A Word on Testing

A Different Look at Passing Parameters and Returning Values

Examples to "act out" in Class

Worker 0:
- clap, clap, clap!

Worker 1:
- get a number
- ***return*** that number multiplied by 2, plus 1.

Worker 2:
- get a number
- ***return*** that number modulo 10

Examples to "act out" in Class

Worker 3:

- get a number
- compute that number multiplied by 2 and incremented by 1
- ***return*** the result modulo 10

Python Version

ActOut.py - /Users/thiebaut/Desktop/Dropbox/111/ActOut.py (3.5.4)

```
# ActingOut.py
# D. Thiebaut

def worker0():
    print( "clap, clap, clap!" )

def worker1( num1 ):
    num2 = num1 * 2 + 1
    return num2

def worker2( num1 ):
    num2 = num1 % 10
    return num2

def worker3( num1 ):
    num2 = worker1( num1 )
    num3 = worker2( num2 )
    return num3

def main():
    n = 12
    print( "worker1(", n, ")=", worker1( n ) )
    print( "worker2(", n, ")=", worker2( n ) )
    print( "worker3(", n, ")=", worker3( n ) )

main()
```

Ln: 16 Col: 11

Python Version

ActOut.py - /Users/thiebaut/Desktop/Dropbox/111/ActOut.py (3.5.4)

```
# ActingOut.py
# D. Thiebaut

def worker0():
    print( "clap, clap, clap!" )

def worker1( num1 ):
    num2 = num1 * 2 + 1
    return num2

def worker2( num1 ):
    num2 = num1 % 10
    return num2

def worker3( num1 ):
    num2 = worker1( num1 )
    num3 = worker2( num2 )
    return num3

def main():
    n = 12
    print( "worker1(", n, ")=", worker1( n ) )
    print( "worker2(", n, ")=", worker2( n ) )
    print( "worker3(", n, ")=", worker3( n ) )

main()
```

Ln: 16 Col: 11

```
>>>
worker1( 12 )= 25
worker2( 12 )= 2
worker3( 12 )= 5
>>>
```

Return and Print are very different statements!

Review

Line-Up your Minions!

Acting Out a Program

Functions Returning Values

Exercises

A Word on Testing

A Different Look at Passing Parameters and Returning Values

Function Boot-Camp

Exercise 1:

- Write a function that gets a temperature in Celsius as a parameter and returns the equivalent Fahrenheit.

$$F = C * 9 / 5 + 32$$

- Test the function from the Python shell

Function Boot-Camp

Exercise 2:

- Write a function that is given a string in the form dd MMM yyyy, and that ***returns*** it as mmddyyyy

Example: 7 Oct 2015 —> 10072015

Formatting Trick: Leading Zeros

Python 3.5.0b1 Shell

```
Python 3.5.0b1 (v3.5.0b1:071fefbb5e3d, May 23 2015, 18:22:54)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> for n in [0, 1, 5, 10, 20, 99]:
        print( "{0:02}".format( n ) )

00
01
05
10
20
99
>>>
```

Ln: 14 Col: 4

Review

Line-Up your Minions!

Acting Out a Program

Functions Returning Values

Exercises

Scope

A Word on Testing

A Different Look at Passing Parameters and Returning Values

The Concept of "Scope" of a function

Example

```
def process(  ):

    def fun1( ):
        x = input( "amount? " )
        return x

    def fun2( a, b ):
        x = a // b
        y = a % b
        return x, y


    amount = fun1()
    No12, left = fun2( amount, 12 )
    print( No12 )

def main():
    print( "welcome" )

    process()

main()
```

Example

```
def process(  ):

    def fun1( ):
        x = input( "amount? " )
        return x

    def fun2( a, b ):
        x = a // b
        y = a % b
        return x, y


    amount = fun1()
    No12, left = fun2( amount, 12 )
    print( No12 )

def main():
    print( "welcome" )

    process()

main()
```

Example: Why is it bad?

```
def process(  ):

    def fun1( ):
        x = input( "amount? " )
        return x

    def fun2( a, b ):
        x = a // b
        y = a % b
        return x, y

    amount = fun1()
    No12, left = fun2( amount, 12 )
    print( No12 )

def main():
    print( "welcome" )

    process()

main()
```

Example: Why is it bad?

```
def process(  ):
    def fun1( ):
        x = input( "amount? " )
        return x

    def fun2( a, b ):
        x = a // b
        y = a % b
        return x, y

    amount = fun1()
    No12, left = fun2( amount, 12 )
    print( No12 )

def main():
    print( "welcome" )

    process()

main()
```

Scope of process()

Example: Why is it bad?

```
def process(  ):
    def fun1( ):
        x = input( "amount? " )
        return x

    def fun2( a, b ):
        x = a // b
        y = a % b
        return x, y

    amount = fun1()
    No12, left = fun2( amount, 12 )
    print( No12 )
def main():
    print( "welcome" )

    process()
    x = fun1()

main()
```

Scope of process()

Example: Why is it bad?

```
def process(  ):
    def fun1( ):
        x = input( "amount? " )
        return x

    def fun2( a, b ):
        x = a // b
        y = a % b
        return x, y

    amount = fun1()
    No12, left = fun2( amount, 12 )
    print( No12 )
def main():
    print( "welcome" )

    process()
    x = fun1()

main()
```

Scope of process()

ERROR!

How to Fix it?

```
def process(  ):

    def fun1( ):
        x = input( "amount? " )
        return x

    def fun2( a, b ):
        x = a // b
        y = a % b
        return x, y


    amount = fun1()
    No12, left = fun2( amount, 12 )
    print( No12 )

def main():
    print( "welcome" )

    process()
    x = fun1()

main()
```

The Right Way

```
def fun1( ):
    x = input( "amount? " )
    return x

def fun2( a, b ):
    x = a // b
    y = a % b
    return x, y

def process(  ):
    amount = fun1()
    No12, left = fun2( amount, 12 )
    print( No12 )

def main():
    print( "welcome" )

    process()
    x = fun1()

main()
```






The Right Way

```
def fun1( ):
    x = input( "amount? " )
    return x

def fun2( a, b ):
    x = a // b
    y = a % b
    return x, y

def process(  ):
    amount = fun1()
    No12, left = fun2( amount, 12 )
    print( No12 )

def main():
    print( "welcome" )

    process()
    x = fun1()

main()
```

Review

Line-Up your Minions!

Acting Out a Program

Functions Returning Values

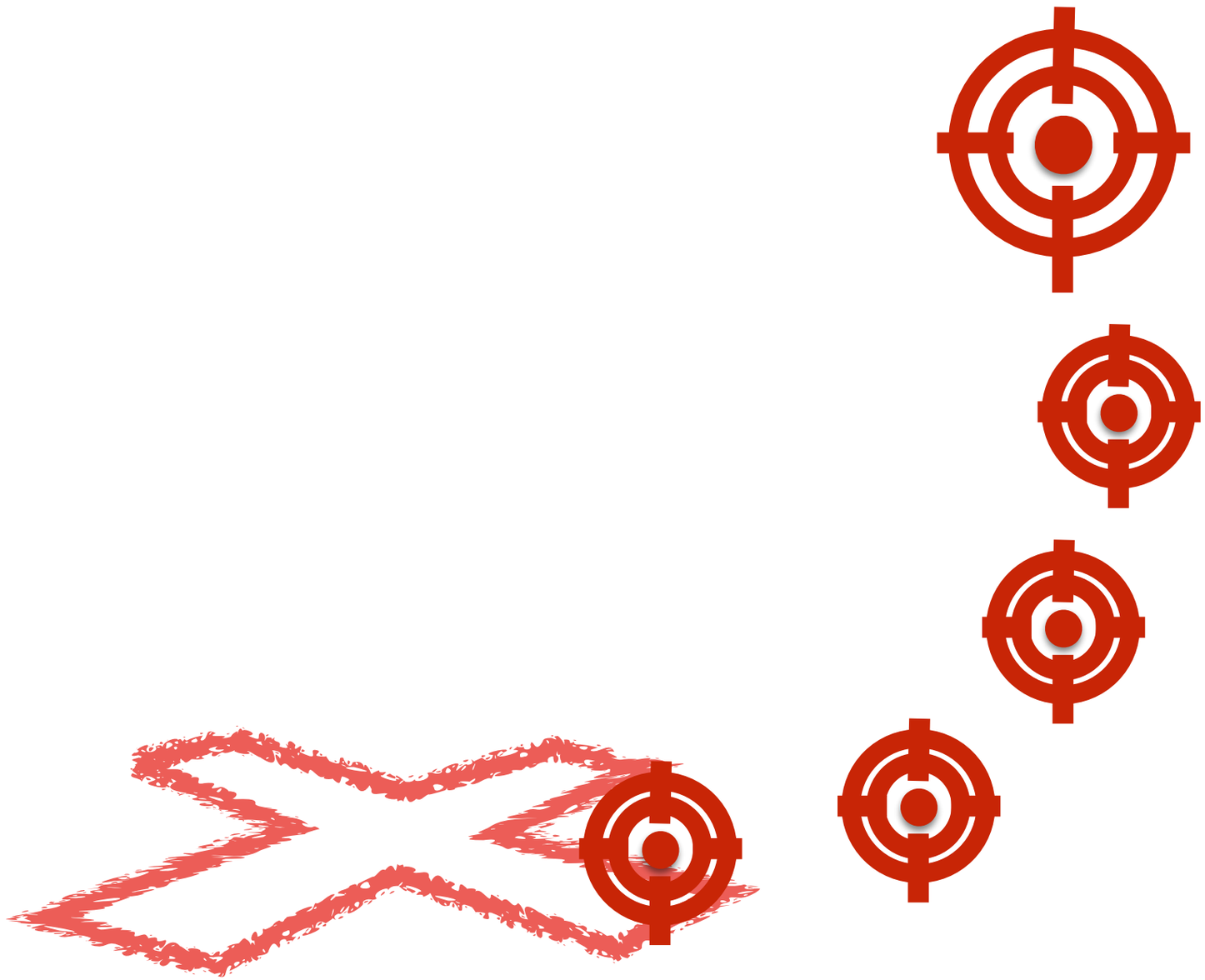
Exercises

Scope

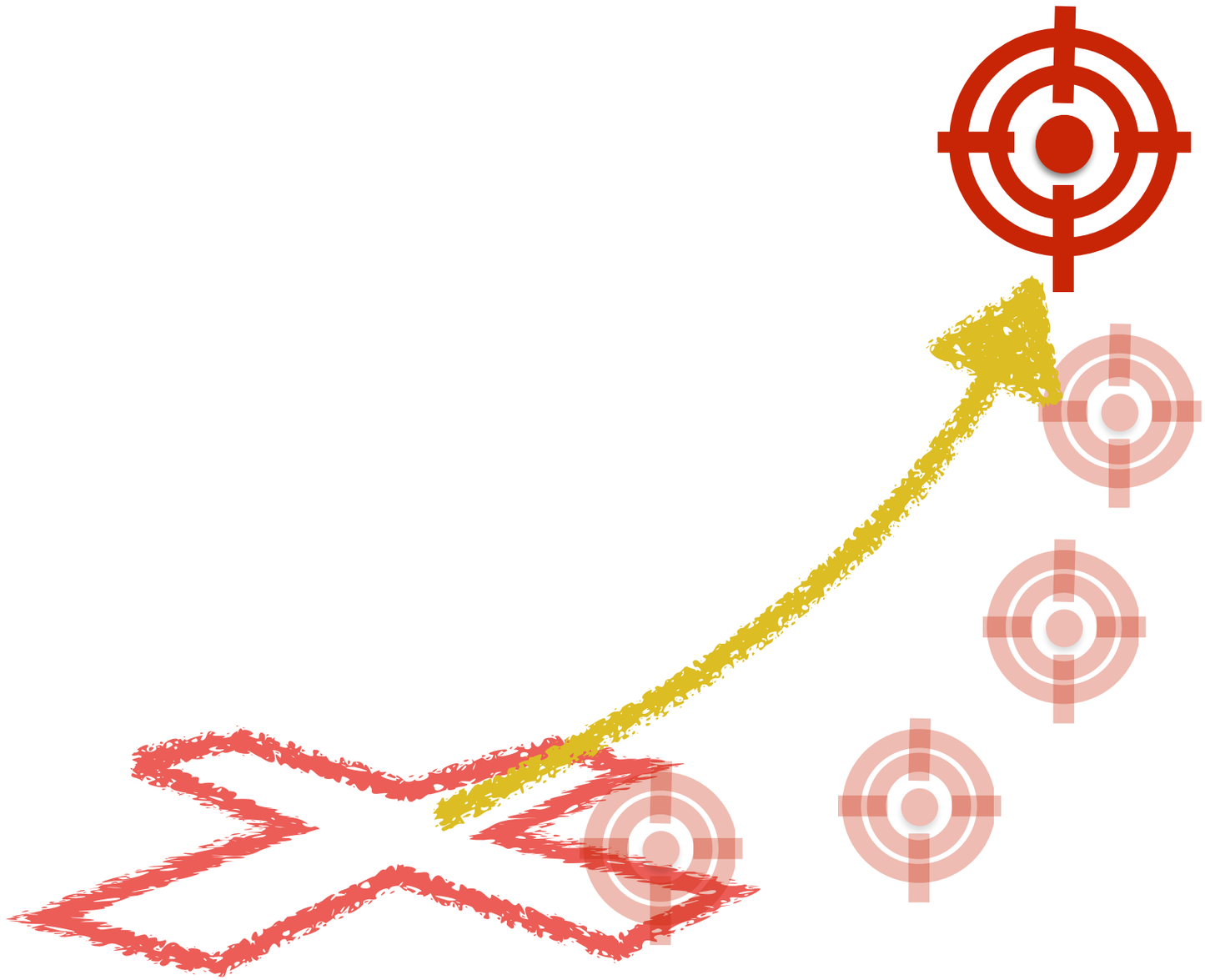
A Word on Testing

A Different Look at Passing Parameters and Returning Values

A Few Words On Testing…

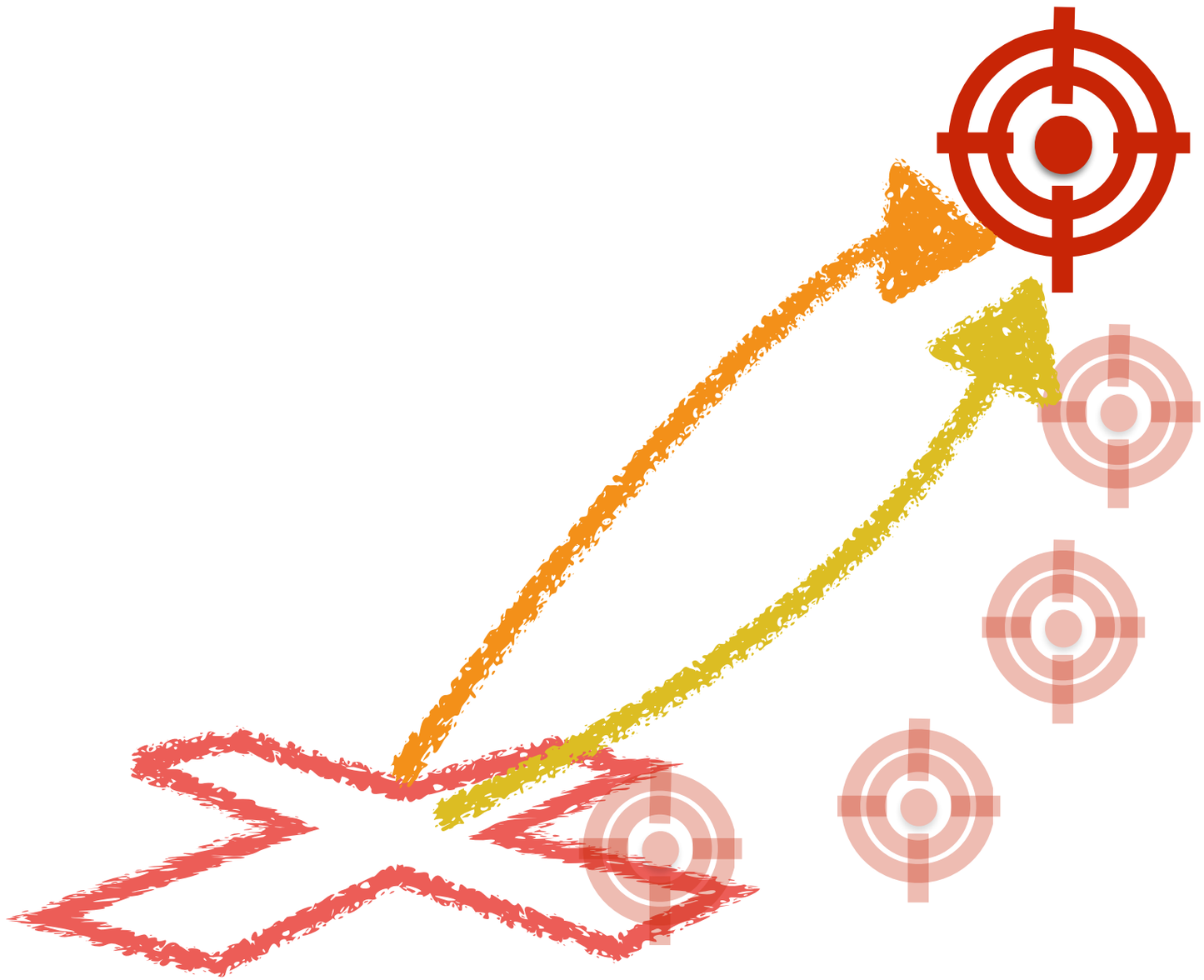


Test #1



Test #1

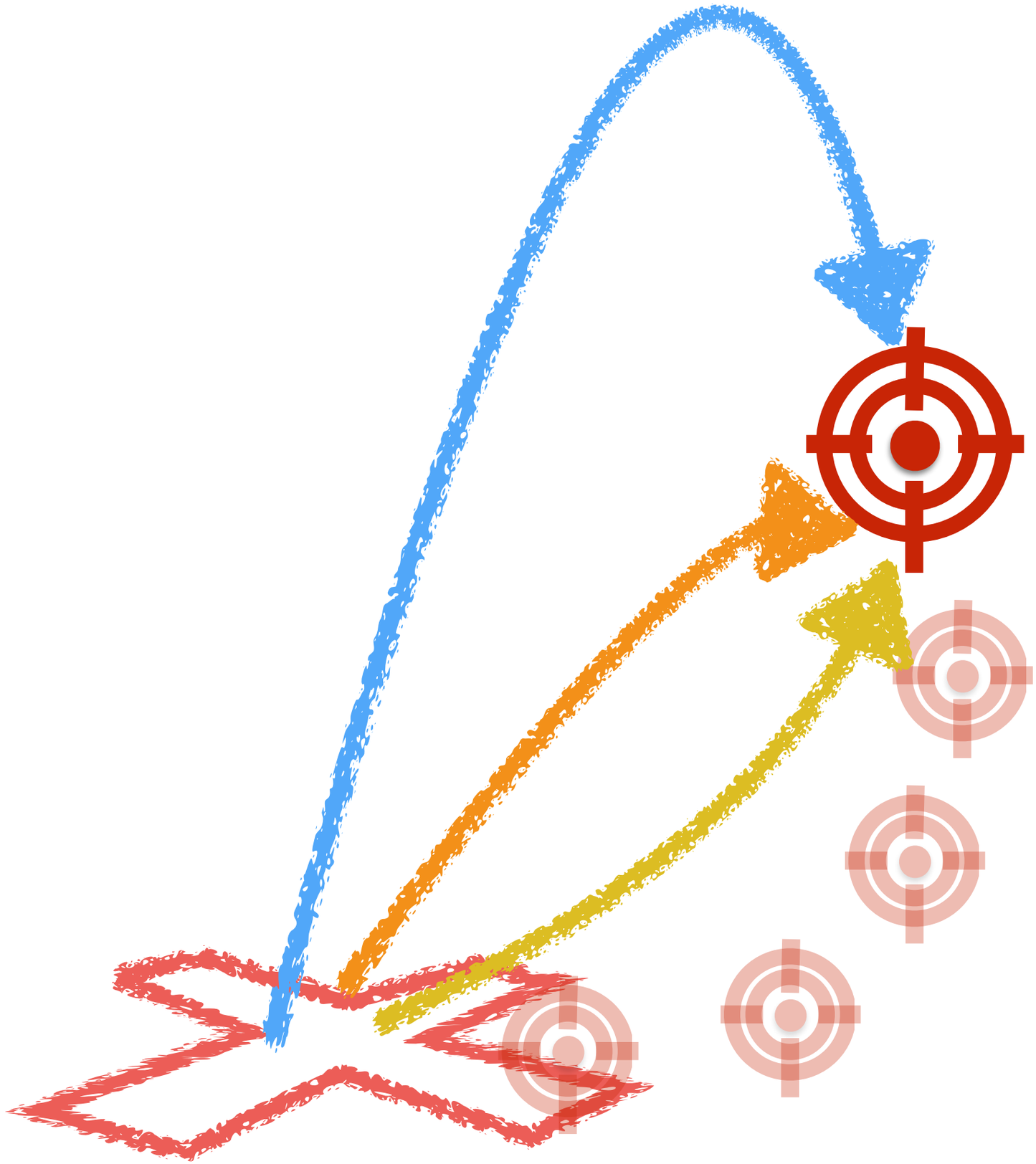
Test #2



Test #1

Test #2

Test #3



Test #1

Test #2

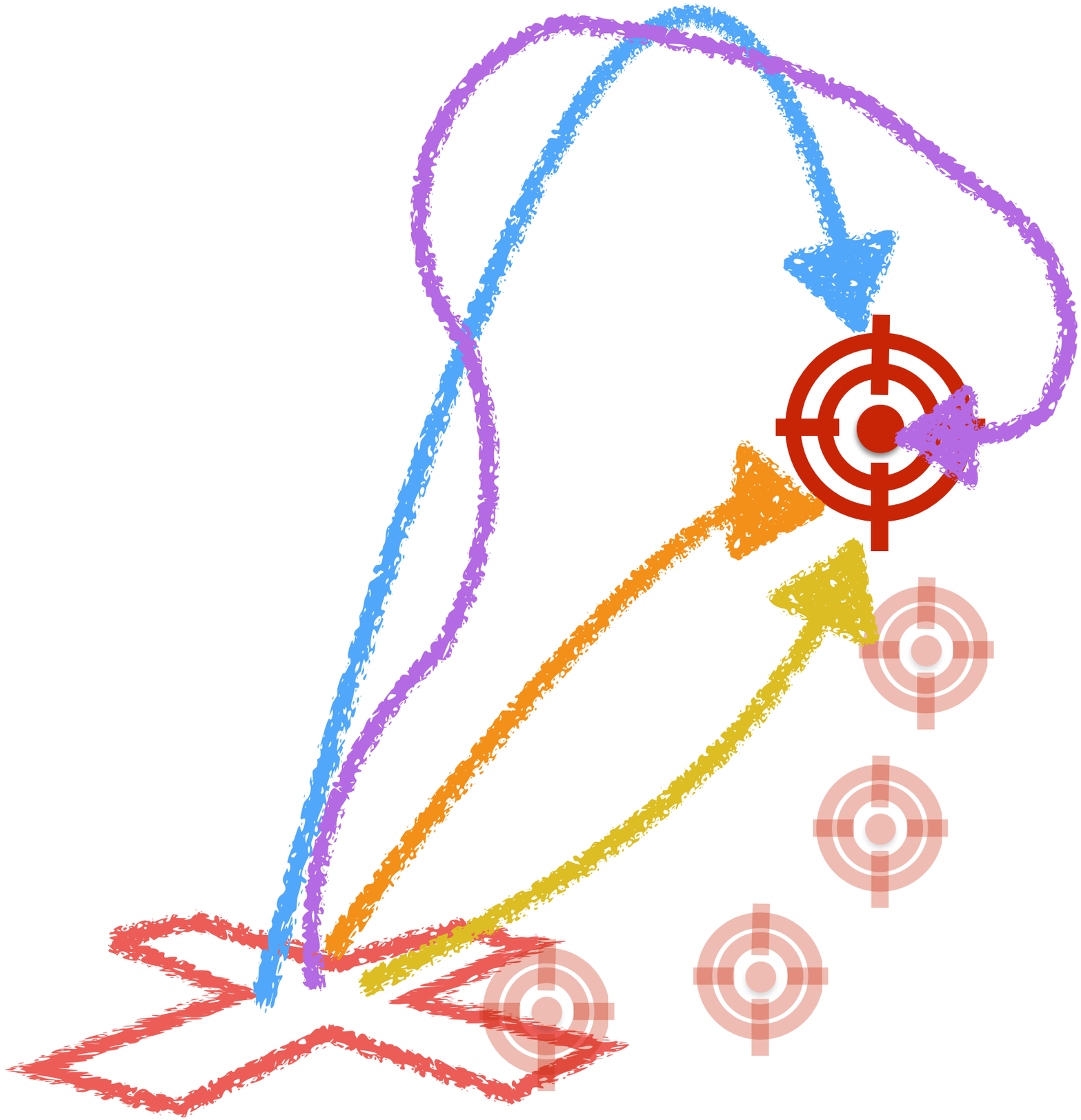
Test #3

.

.

.

Test #n



Test #1

Test #2

Test #3

.

.

.

Test #n

Robustness

- **Inputting numbers**:
  - **Test special values: 0, 1, -1, negative, very large.**
- **Inputting lists**:
  - **Test empty lists: [] , or ()**
- **Inputting strings:**
  - **Mix upper- & lower-case: "HEllo"**
  - **Empty string: ""**
  - **Add spaces: " Hello Kitty! "**

Review

Line-Up your Minions!

Acting Out a Program

Functions Returning Values

Exercises

Scope

A Word on Testing

A Different Look at Passing Parameters and Returning Values

Calling Functions

Pass (Parameters)

Calling Functions

Do some work

Calling Functions

Return value(s)

```
def minion( a, b ):
    x = a // b
    y = a % b
    return y

def Gru():
  x = minion( 5, 3)
  print( x )

Gru()
```

Pass Parameter(s)

```
def minion( a, b ):
    x = a // b
    y = a % b
    return y

def Gru():
  x = minion( 5, 3)
  print( x )

Gru()
```

Do some work

Calling F

```
def minion( a, b ):
    x = a // b
    y = a % b
    return y

def Gru():
   x = minion( 5, 3)
   print( x )

Gru()
```

Return value(s)

We stopped here last time…

Function Boot Camp

Exercise:

- Revisit the Teller Machine program
  - With one function that breaks down the amount in number of $-xx and left over
  - With one function that returns ALL the number of $-xx bills

Function Boot Camp

Exercise:

- Write a function that gets a string of DNA symbols, locates the ***marker*** "ACA," and returns only the DNA string following the marker. The marker is **always** in the DNA.
- Make the function more generic so that it gets the DNA string **and** the marker as parameters.

Function Boot Camp

Write a program that will censure text by blocking out specific words. The text is stored in a text file called **memo.txt**. The words to be removed are stored in a file called **badwords.txt.** The censured message is printed on the screen.

memo.txt:

We need 10 tons of burgers and 1 ton of ketchup sent to the President before midnight.

badwords.txt:

burger
ketchup
president

Function Boot Camp

Write a program that will censure text by blocking out specific words. The text is stored in a text file called **memo.txt**. The words to be removed are stored in a file called **badwords.txt.** The censured message is printed on the screen.

memo.txt:

We need 10 tons of burgers and 1 ton of ketchup sent to the President before midnight.

badwords.txt:

burger
ketchup
president

```
We need 10 tons of xxxxxxx and 1 ton of xxxxxxx
sent to the xxxxxxxxx before midnight.
```