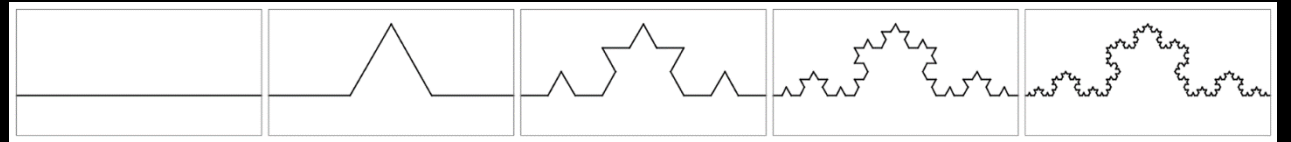


# Intro to Creative Computing

Week 12

# This class

- Recursion
- Exceptions
- Our assignment – planning out a pseudocode



# Recursion

- Code inside a function calling the function itself
- Can be understood (and also rewritten) as calling a function in loop while sending some data with it
- Good example can be found in math formulas!

# Recursion - example

- Factorial:
  - (def) In mathematics, the factorial of a positive integer  $n$ , denoted by  $n!$ , is the product of all positive integers less than or equal to  $n$ .

$$n! = n \times (n - 1) \times (n - 2) \times (n - 3) \times \cdots \times 3 \times 2 \times 1.$$

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120.$$

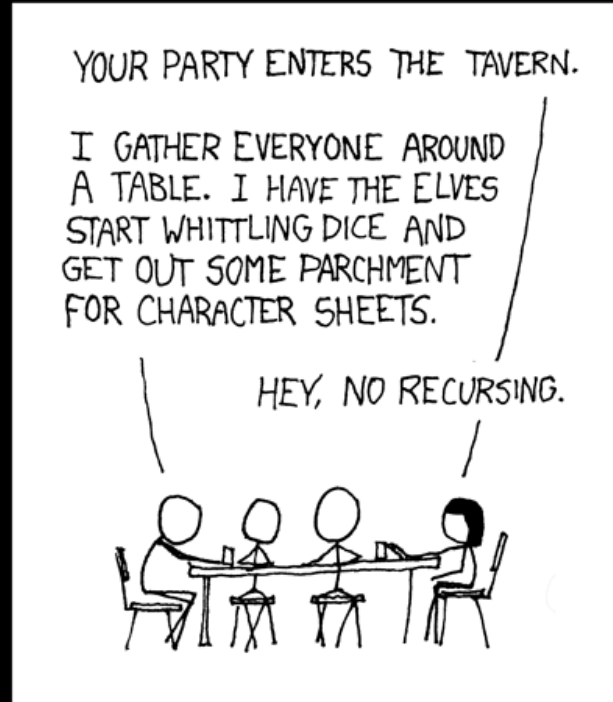
# Factorial

- Factorial:
  - For a number  $n$  we want the result to be  $n \cdot$  (smaller by 1 and repeat)
  - So factorial of  $n$  is:  $n$  times factorial of  $(n-1)$

$$n! = n \cdot (n - 1)!$$

- **Recursive definition** (explain *factorial* by using *factorial*)
- Our **end condition** is 1 (or 0 and we define that  $0! = 1$ )

recursive call



end condition

<https://www.xkcd.com/1739/>

# Factorial in code

- **Recursive definition** (explain *factorial* by using *factorial*)

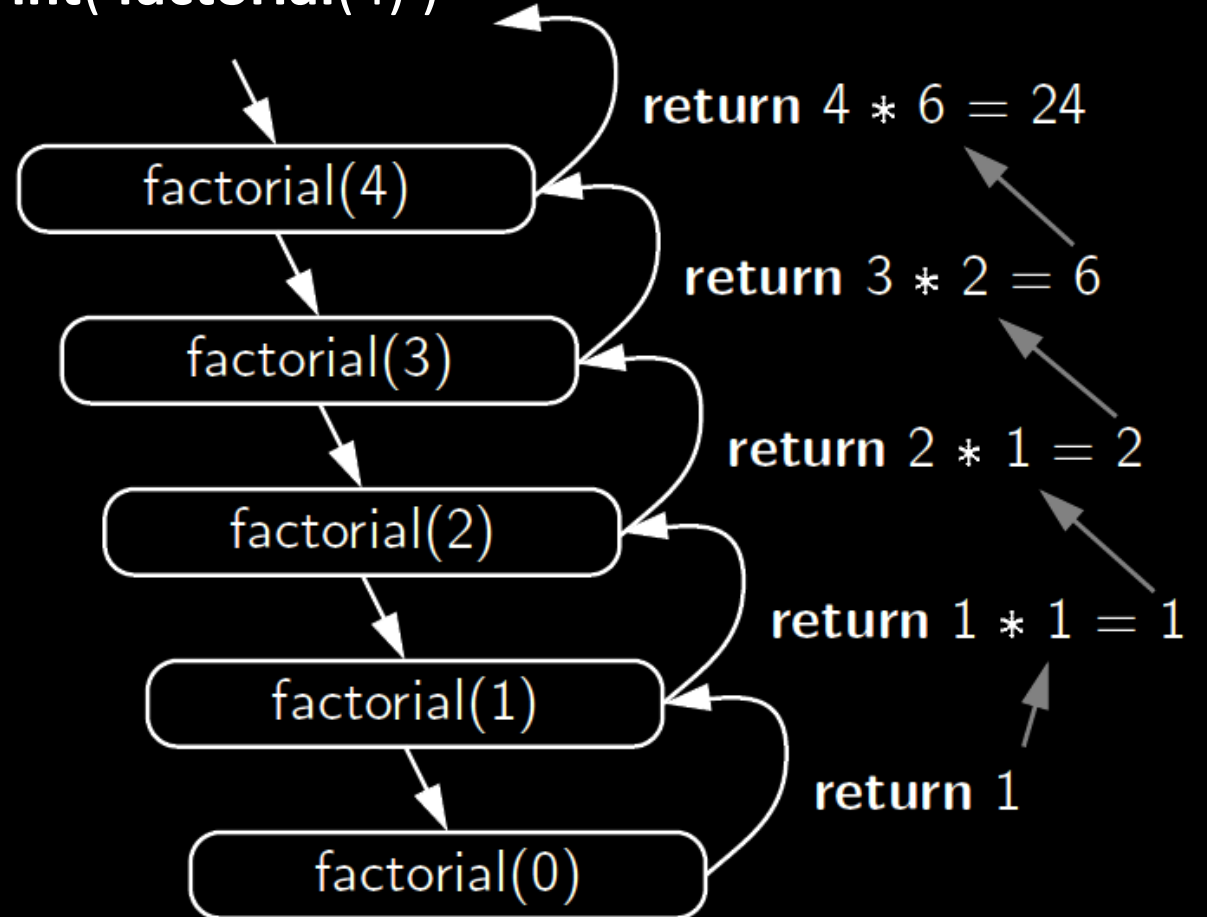
$$n! = n \cdot (n - 1)!$$

- Our **end condition** is 0 (we define that  $0! = 1$ )

```
def factorial(n):  
    # stop condition (otherwise we run forever!)  
    if n == 0:  
        return 1  
  
    else:  
        # do something with the data  
        return n * factorial(n - 1) # n * (n - 1)!
```

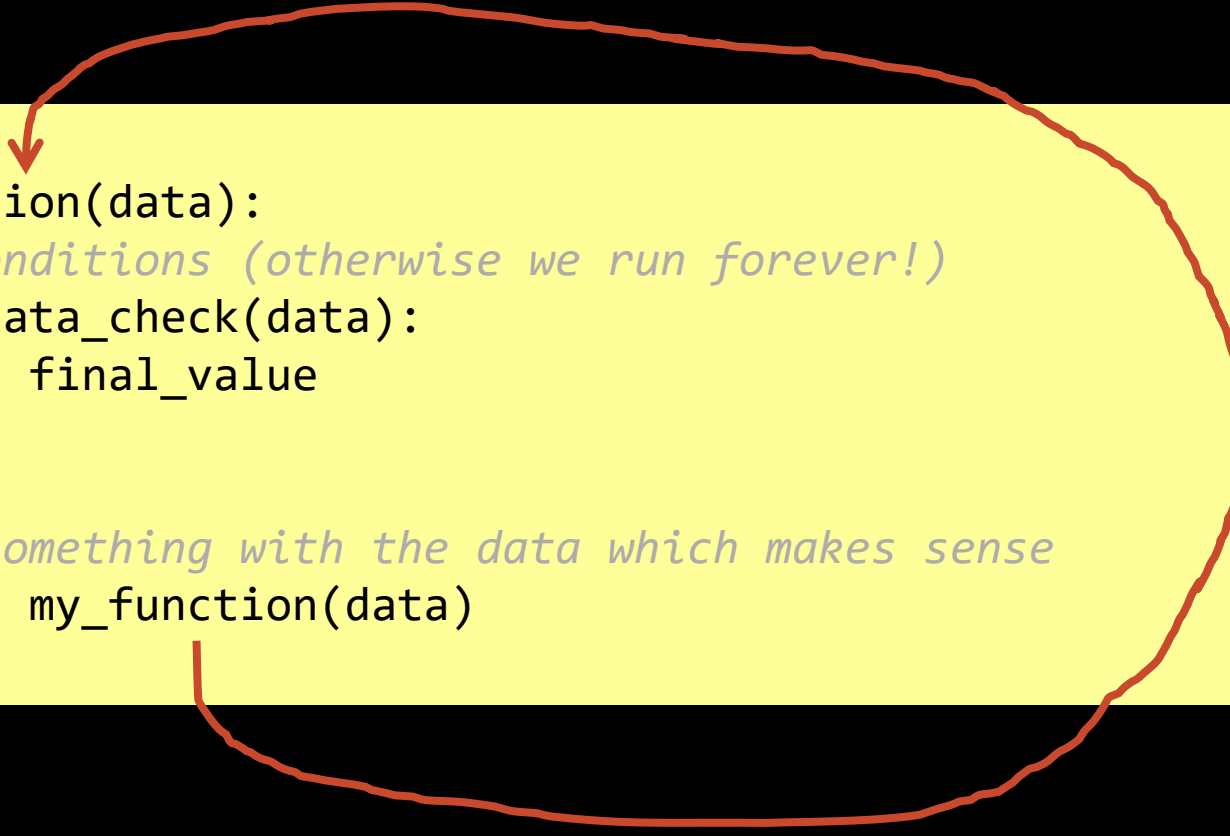
```
def factorial(n):  
    # stop condition  
    if n == 0:  
        return 1  
  
    else:  
        # do something with the data  
        return n * factorial(n - 1)
```

**print( factorial(4) )**

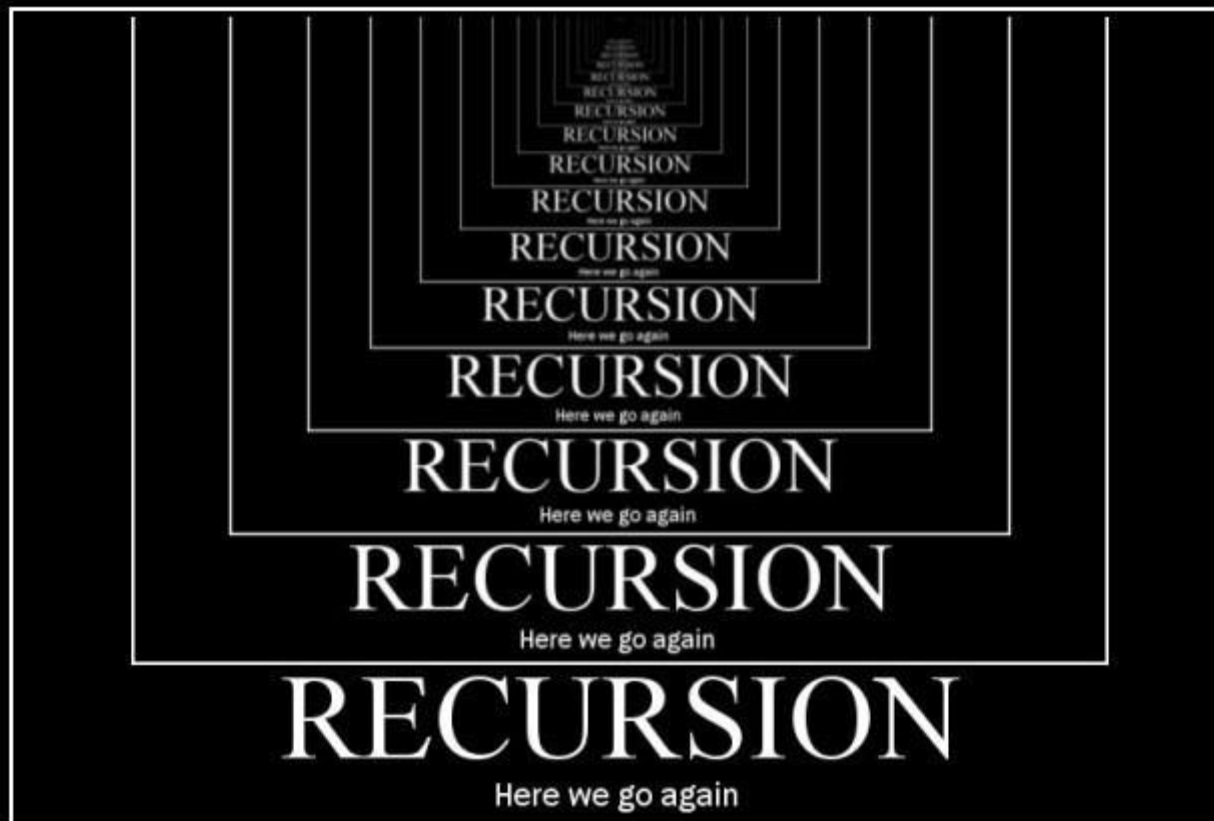




# Recursion – general form



```
def my_function(data):  
    # stop conditions (otherwise we run forever!)  
    if some_data_check(data):  
        return final_value  
  
    else:  
        # do something with the data which makes sense  
        return my_function(data)
```



RECURSION  
Here we go again

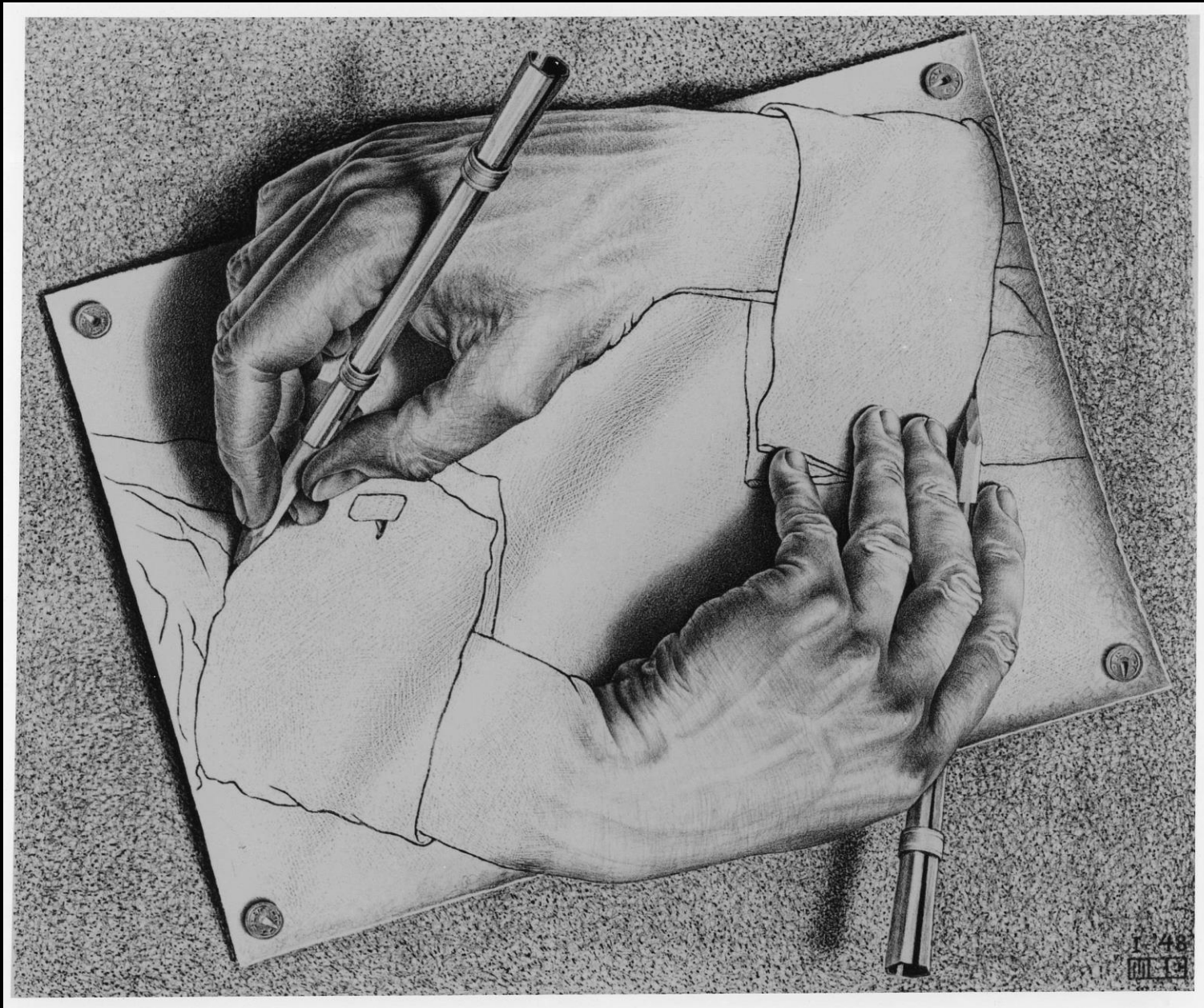


<https://www.xkcd.com/1739/>

# Recursions used in the arts?

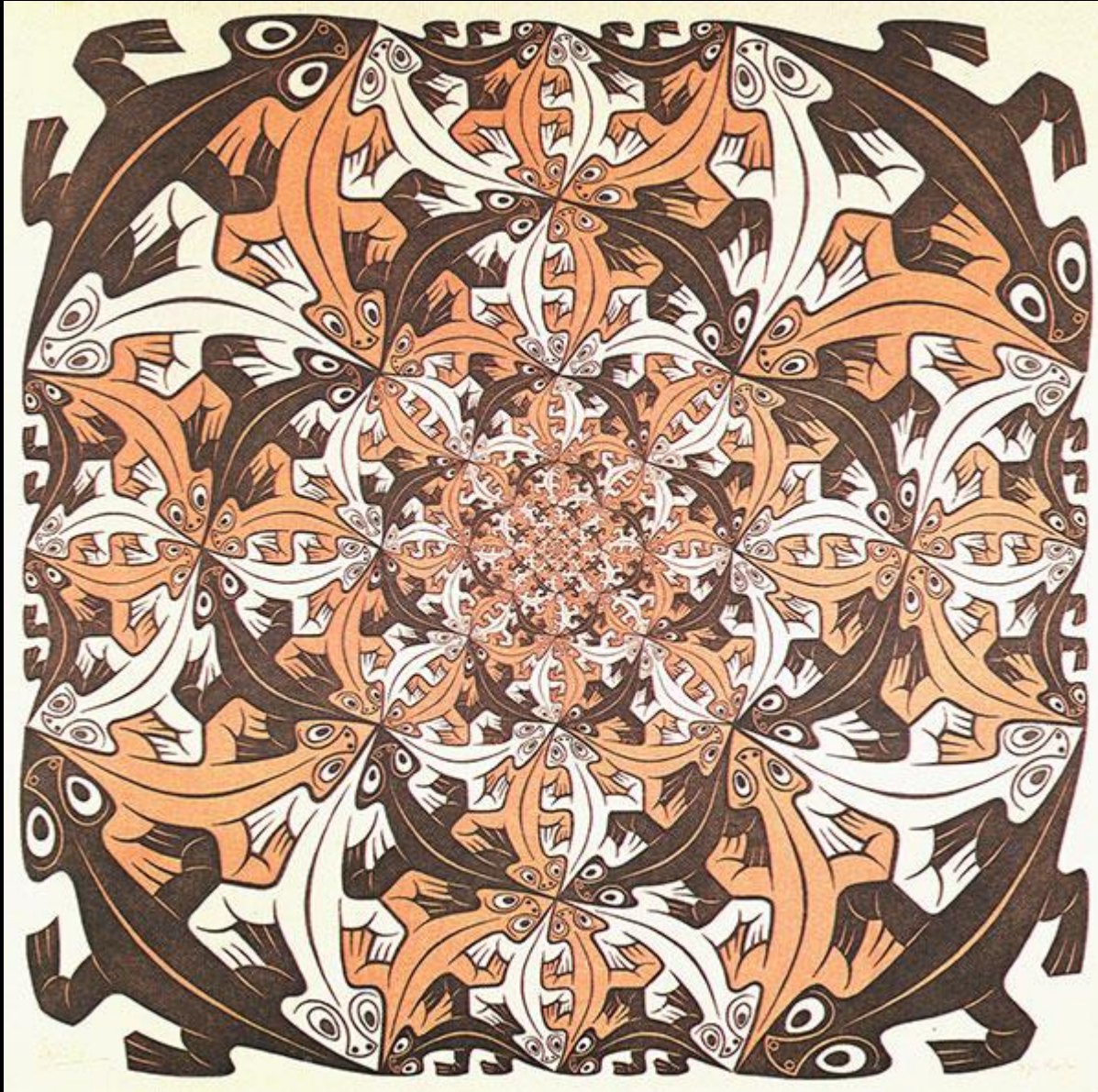
- Maurits Cornelis Escher
- Inspiration from Islamic architecture patterns in Granada and Cordoba





M. C. Escher – Drawing hands





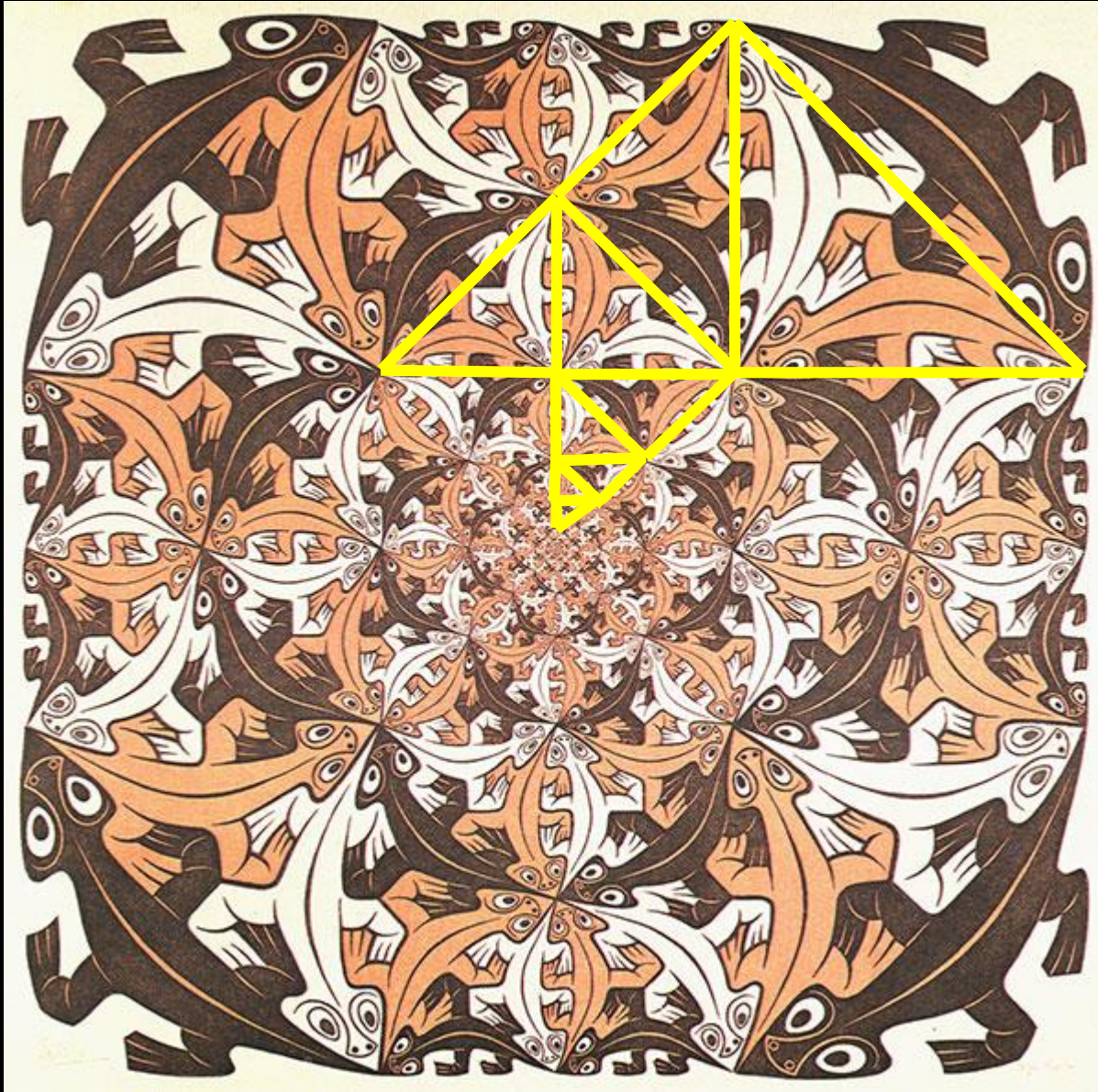
### **Tessellation**

- (def) an arrangement of shapes closely fitted together, especially of polygons in a repeated pattern without gaps or overlapping
- Regular repetition

### **Recursion (also Self-similarity)**

- Recursive repetition





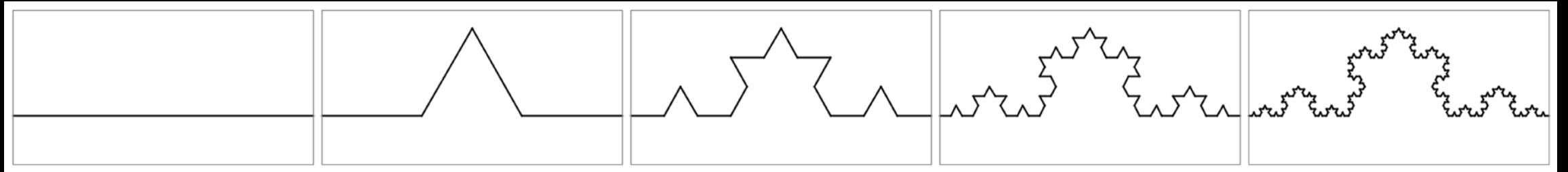
### **Tessellation**

- (def) an arrangement of shapes closely fitted together, especially of polygons in a repeated pattern without gaps or overlapping
- Regular repetition

### **Recursion (also Self-similarity)**

- Recursive repetition

# Fractals

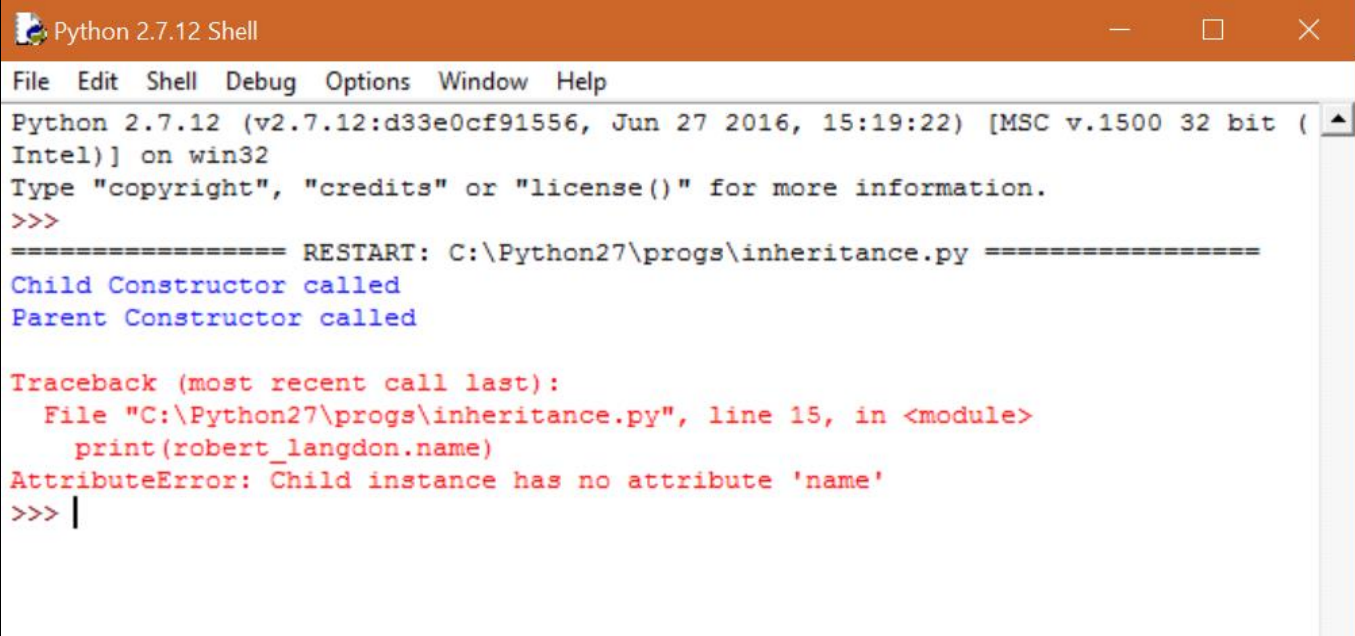


- More in the maths class “19/20 Coding Two: Data, Maths and Methods”
- Sneak peak: <https://natureofcode.com/book/chapter-8-fractals/>



# Exceptions

- We often encounter this:



```
Python 2.7.12 Shell
File Edit Shell Debug Options Window Help
Python 2.7.12 (v2.7.12:d33e0cf91556, Jun 27 2016, 15:19:22) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Python27\progs\inheritance.py =====
Child Constructor called
Parent Constructor called

Traceback (most recent call last):
  File "C:\Python27\progs\inheritance.py", line 15, in <module>
    print(robert_langdon.name)
AttributeError: Child instance has no attribute 'name'
>>> |
```

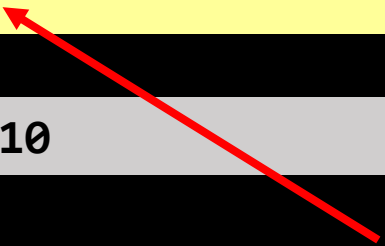
- Exceptions are a way how to react to Errors inside the program

# Exceptions - example

- We want to get a number from the user:

```
x = int(input("Please enter a number: "))
```

```
Please enter a number: 10
```



Any idea when this might  
break???

# Exceptions - example

- We want to get a number from the user:

```
x = int(input("Please enter a number: "))
```

```
Please enter a number: 10
```

```
Please enter a number: sadasd
```

```
Traceback (most recent call last):
```

```
  File "main.py", line 3, in <module>
```

```
    x = int(input("Please enter a number: "))
```

```
ValueError: invalid literal for int() with base 10: 'sadasd'
```

# Exceptions - example

- If we expect difficulties, we can put the code in the try – except blocks:

```
try:
    # code where we expect to see errors:
    x = int(input("Please enter a number: "))

except:
    # we caught an error!
    print("Something went wrong!!!")
    x = -1

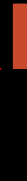
print("x =", x)
```

Run  
without  
errors



x = 10 (for example)

Run with error  
inside the try  
block



x = -1 (or pick a default value)

Error stops it and  
jumps to the  
“except” block

# Exceptions - example

- Full example, if we want to get an integer from the user:

```
while True:
    try:
        # code where we expect to see errors:
        x = int(input("Please enter a number: "))
        break
    except:
        # we caught an error!
        print("This is not a number! Please try again...")
```

# Exceptions

- Some more details in the reading at:
  - <https://runestone.academy/runestone/books/published/thinkcspy/Exceptions/toctree.html>

Pause 1

# Marking!

- *Assignment: Element 1 - Course Work (50%)*
  - 25% - Program you submit (two-player game)
    - <https://moodle.arts.ac.uk/course/view.php?id=38145>
  - 25% - Attendance (for the attendance to count, you still have to submit the program)
  - Deadline for submission: **20.1.2020**
- *Assignment: Element 2 - Exam (50%)*
  - This will be in the last class, on the **21.1.2020**



# Assignment

- Write a small two-player text game where the first player enters a list of 5 things or people they love most, and the second player tries to guess what those things are.

1: Greet the players and prints a description of how to play.

2: Prompt user input for 5 different things or people, then receive that input and saves it. Input each item separately. Then ask the player to input a clue for each item and saves the clue so that it corresponds to the item. Try figuring out how to hide the player's input in your console, or just print a bunch of spaces to hide it!

3: Offer clues to the second player and prompt the player to guess! Go one item at a time. Print out the associated clue and prompt the user to guess the item. Keep a score for how many times the player guesses wrong for each item. When a player gets an item, print a congratulations message. At the end of the whole list, start back at the beginning until all of the items are guessed. If an item has already been guessed correctly, skip that item and move on to the next. Do this for however many turns it takes for the player to guess all of the items. At the end of the game, print the score. Save the score to a file as the best high score.

4: Ask the player if they'd like to replay. If so tell them the all time best high score and replay the game. If not quit the game and tell the player their best score before exiting.

- 1<sup>st</sup> version due 10.1.2020      => mail to: [v.ruzicka@arts.ac.uk](mailto:v.ruzicka@arts.ac.uk)
- final version due 20.1.2020      => moodle!

# Pseudocode - Assignment

- Pseudocode can be used to outline what should your code do in human-legible (*more or less* :)) words:

```
while we don't have enough data:  
    ask for more data and save it  
  
process the data
```

- We don't have to care about syntax or details, only about the functionality

```
Data:  $N_{\text{iterations}}$ , InitialTrainingSet, TestSet,  
        RemainingUnlabeledData,  $N_{\text{add}}$   
TrainingSet = InitialTrainingSet;  
for  $iteration \leftarrow 0$  to  $N_{\text{iterations}}$  do  
    train model(s) on the TrainingSet;  
    evaluate model(s) performance on the TestSet;  
    run the acquisition function on the  
        RemainingUnlabeledData to select top  $N_{\text{add}}$   
        samples;  
    TrainingSet  $\leftarrow$  annotate and add selected  $N_{\text{add}}$   
    data points;  
end  
  
Algorithm 1: Active Learning Algorithm
```

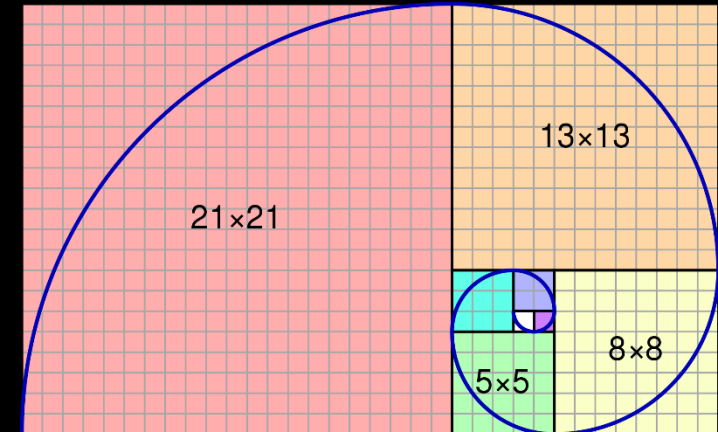
# Pseudocode - Assignment

- We can write our own pseudocode (no rules really) just as a part of program planning – in this case for a task we have all read together

Pause 2

# Tasks

- Task on recursion:
  - Write your own function to calculate Fibonacci numbers:
    - **Recursive formula:**  $fibonacci(n) = fibonacci(n - 1) + fibonacci(n - 2)$
    - **End conditions:**  $fibonacci(0) = 0, fibonacci(1) = 1$
    - The sequence (Fibonacci of 1 to 12) should go as:
      - 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...



- Task on exceptions:
  - Ask the user for a name of a file, try loading it and if it fails write your own error message
  - Bonus: ask the user for a new file name

# Next class

- We will be practicing the basic building blocks of programming...