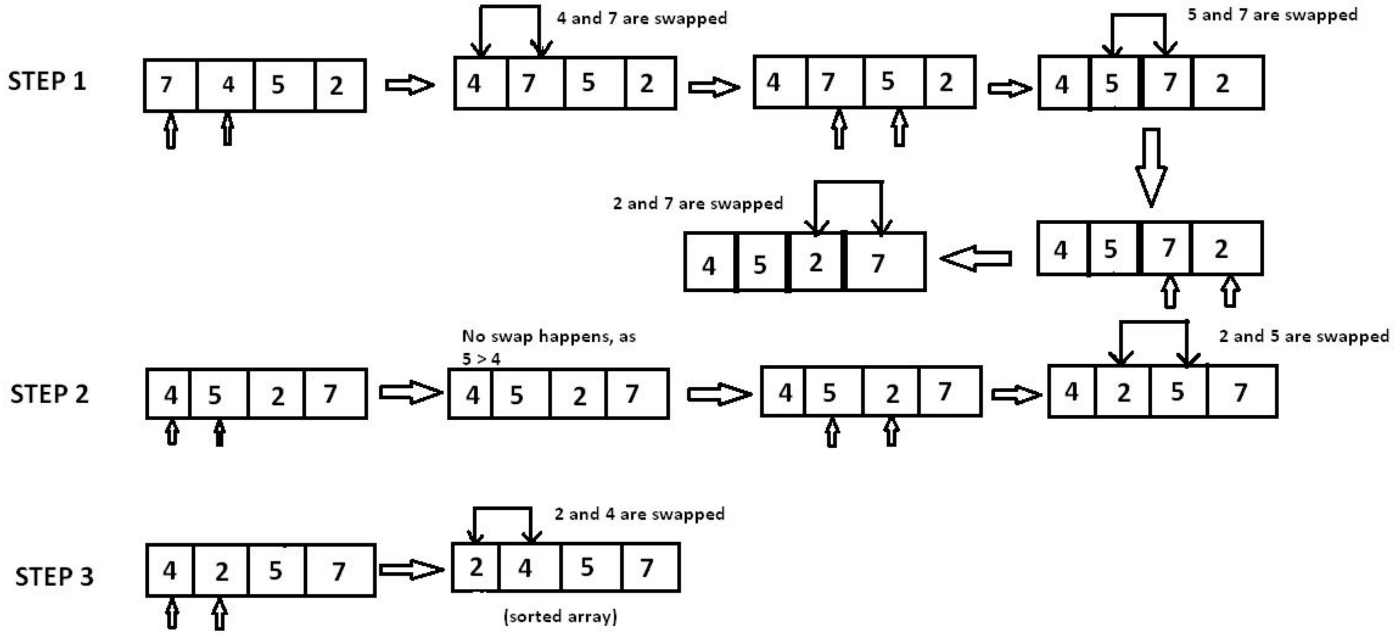
Programming Club Meeting 18 Slides

Review

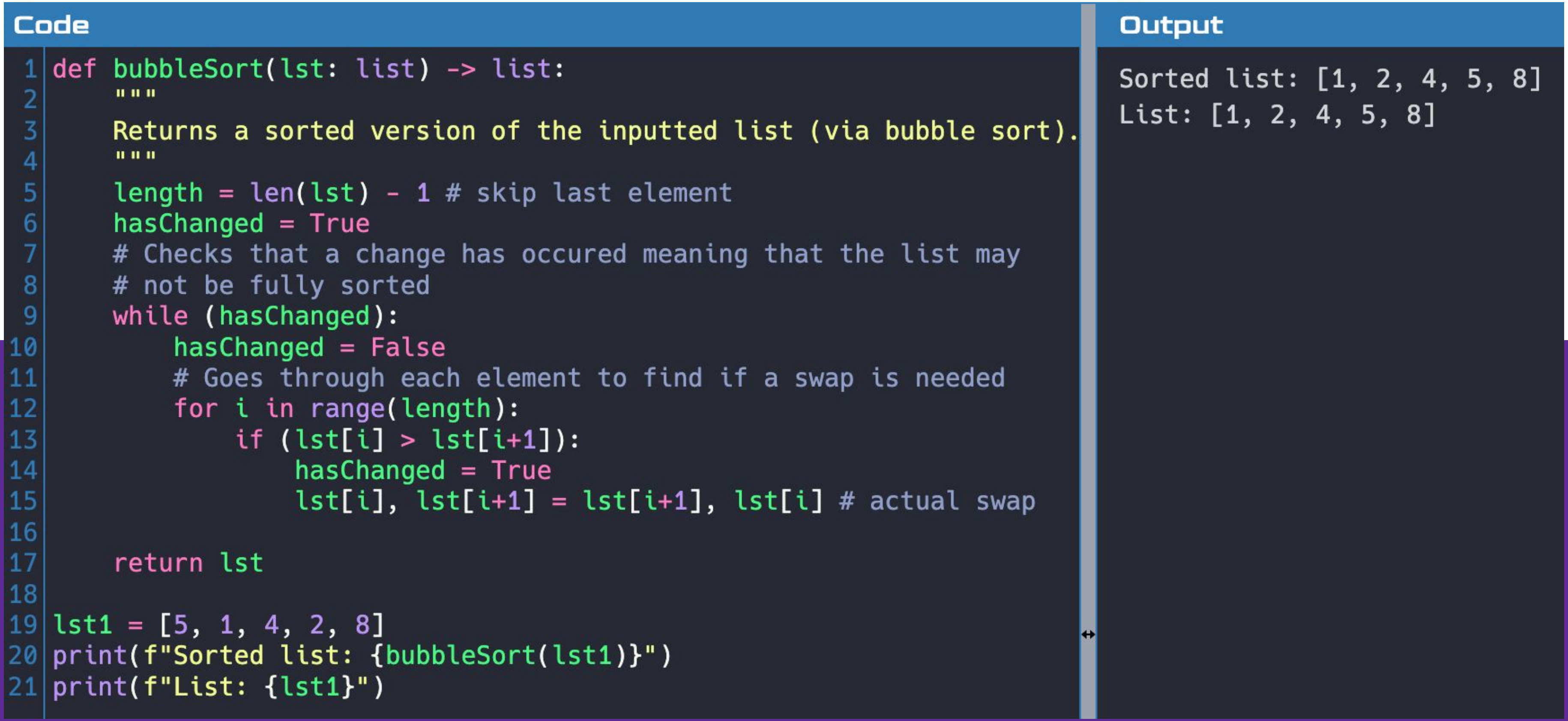
Bubble and Selection Sort

Bubble Sort

- Check 2 consecutive elements, if they’re out of order, swap them
- Repeat until the list is sorted



Bubble Sort


Code

```python
def bubbleSort(lst: list) -> list:
    """
    Returns a sorted version of the inputted list (via bubble sort).
    """
    length = len(lst) - 1 # skip last element
    hasChanged = True
    # Checks that a change has occured meaning that the list may
    # not be fully sorted
    while (hasChanged):
        hasChanged = False
        # Goes through each element to find if a swap is needed
        for i in range(length):
            if (lst[i] > lst[i+1]):
                hasChanged = True
                lst[i], lst[i+1] = lst[i+1], lst[i] # actual swap

    return lst

lst1 = [5, 1, 4, 2, 8]
print(f"Sorted list: {bubbleSort(lst1)}")
print(f"List: {lst1}")
```

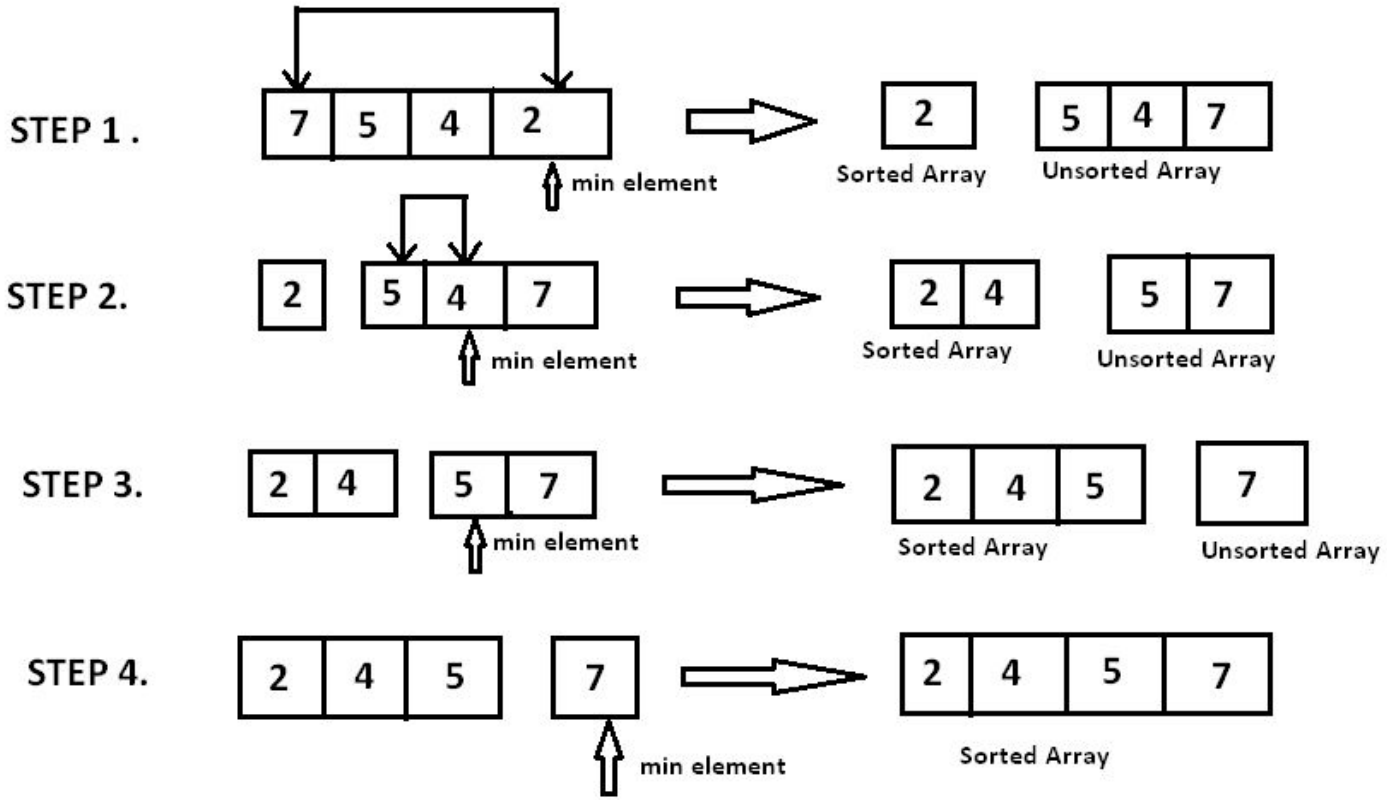
Output

```
Sorted list: [1, 2, 4, 5, 8]
List: [1, 2, 4, 5, 8]
```

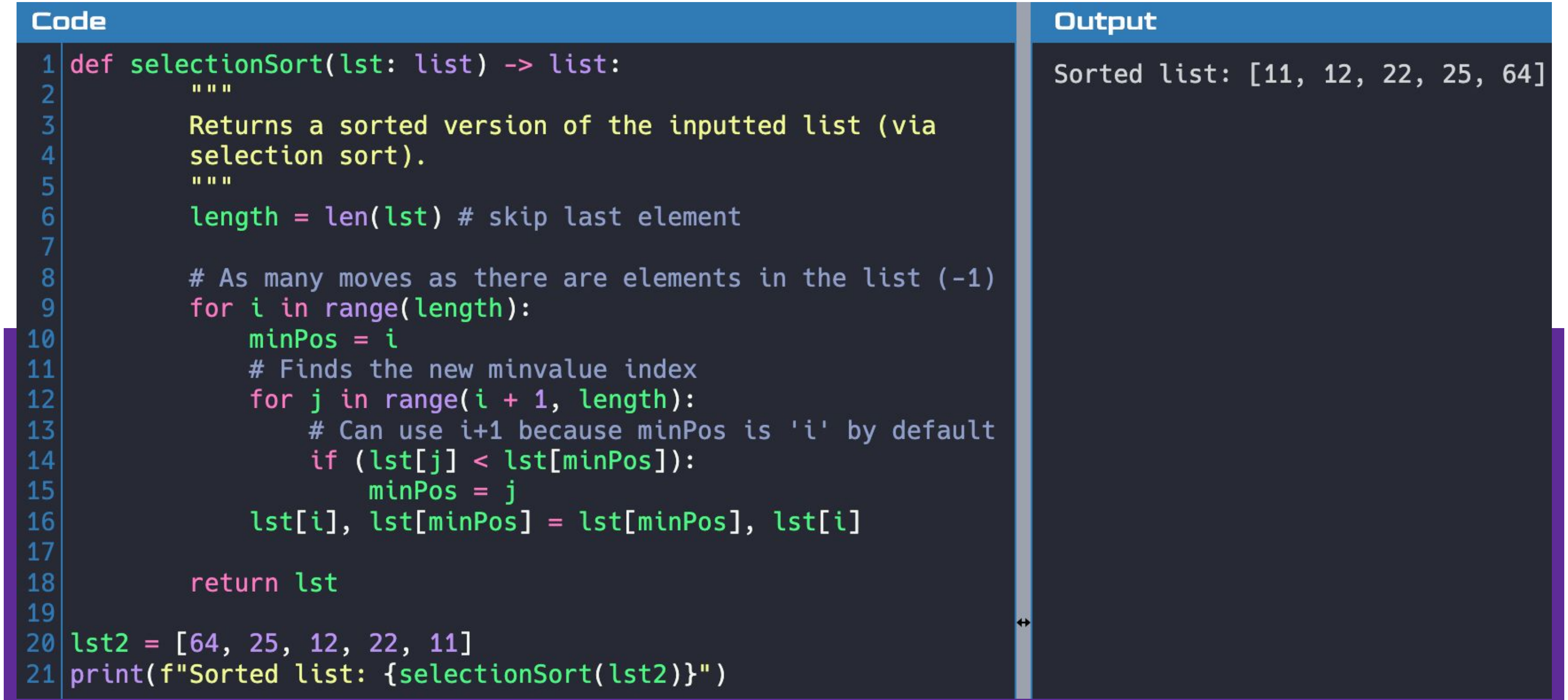
Selection Sort

- Find the minimum unsorted element, move it in front of the rest of the unsorted values
- Repeat until the list is sorted

Selection Sort Visual



Selection Sort



```python
def selectionSort(lst: list) -> list:
        """
        Returns a sorted version of the inputted list (via
        selection sort).
        """
        length = len(lst) # skip last element

        # As many moves as there are elements in the list (-1)
        for i in range(length):
            minPos = i
            # Finds the new minvalue index
            for j in range(i + 1, length):
                # Can use i+1 because minPos is 'i' by default
                if (lst[j] < lst[minPos]):
                    minPos = j
            lst[i], lst[minPos] = lst[minPos], lst[i]

        return lst

lst2 = [64, 25, 12, 22, 11]
print(f"Sorted list: {selectionSort(lst2)}")
```

```
Sorted list: [11, 12, 22, 25, 64]
```

Big O Notation

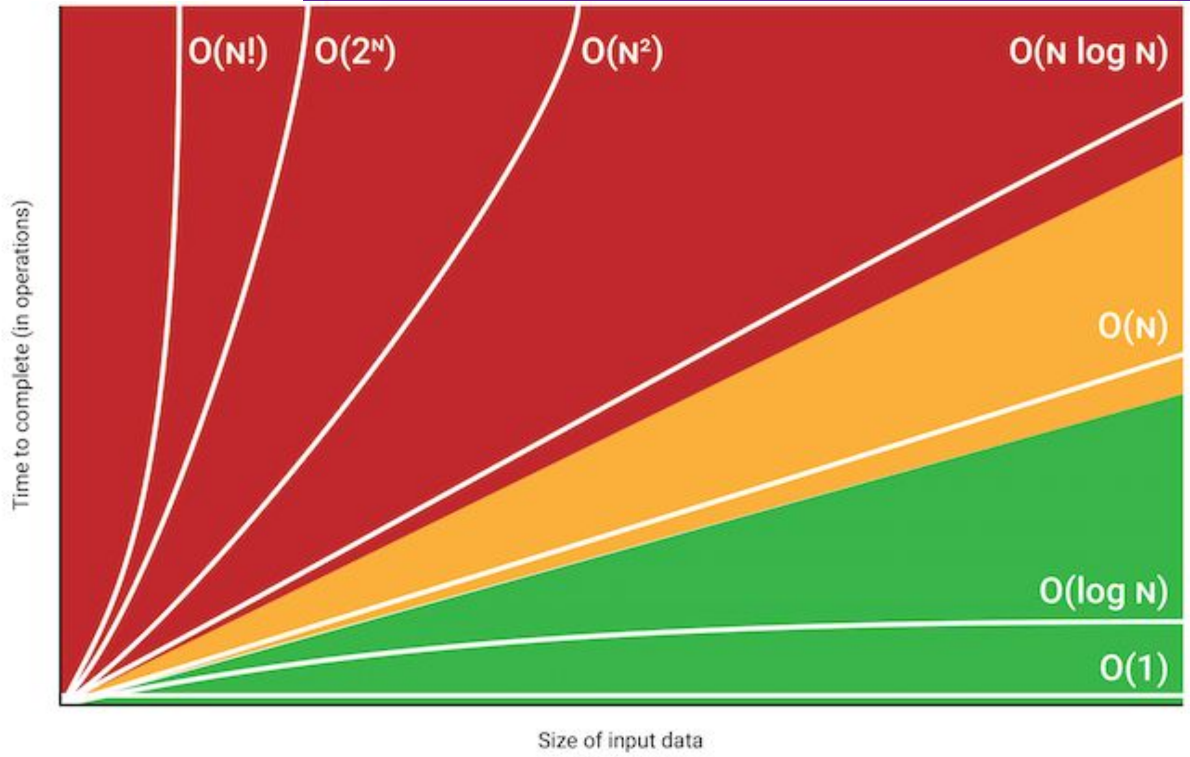
Basics

- Represents how code complexity scales with input size
- Looks like “O(n)”
  - n: input size
- Remove coefficients and lower value n’s
  - $O(3n^2 + n + 5) \rightarrow O(n^2)$

Common Complexities

- O(1) - constant time, ideal but often not possible, usually only for simple algorithms
- O(log n) - logarithmic time, often dive and conquer algorithms, pretty good complexity
- O(n) - linear time, often good complexity, usually happens when going through each element in a list
- O(n^2) - polynomial time, often loops inside of each other, n can be to other powers
  - Both bubble and selection sort are this

Chart

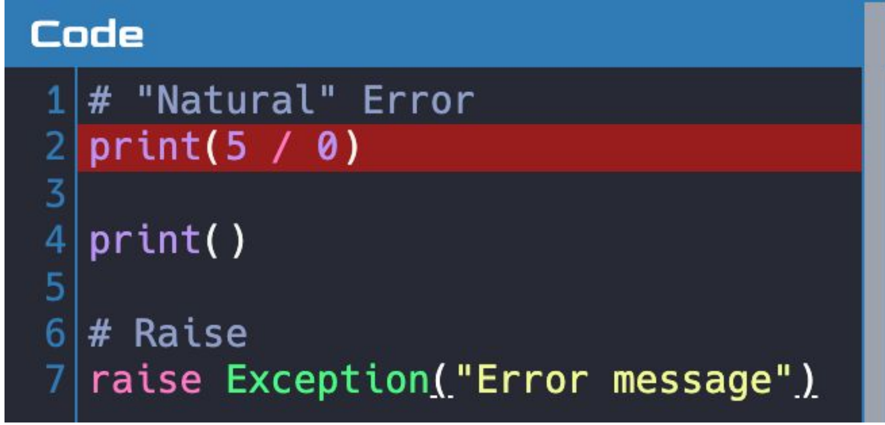


O(N!)
O(2^N)
O(N²)
O(N log N)
O(N)
O(log N)
O(1)
Time to complete (in operations)
Size of input data

Exceptions

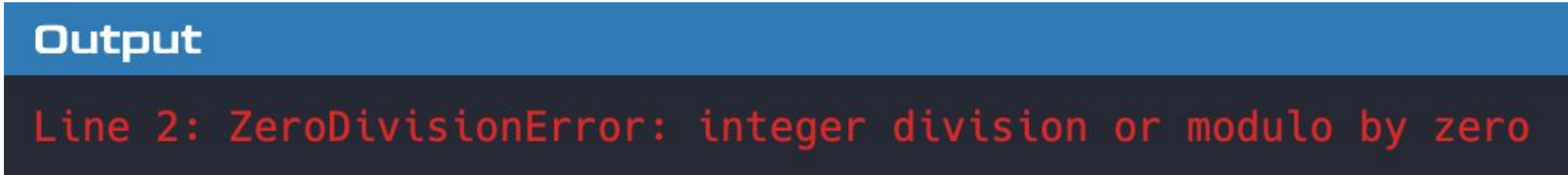
Exceptions

- An exception is when the program runs into an error that makes running the rest of the code impossible
  - “Raised” or “thrown”
- Normally stops the program and outputs an error message
- Many different kinds
  - “Exception” is the base/default one
- Can choose to throw an error with raise



Code

```python
# "Natural" Error
print(5 / 0)

print()

# Raise
raise Exception("Error message")
```



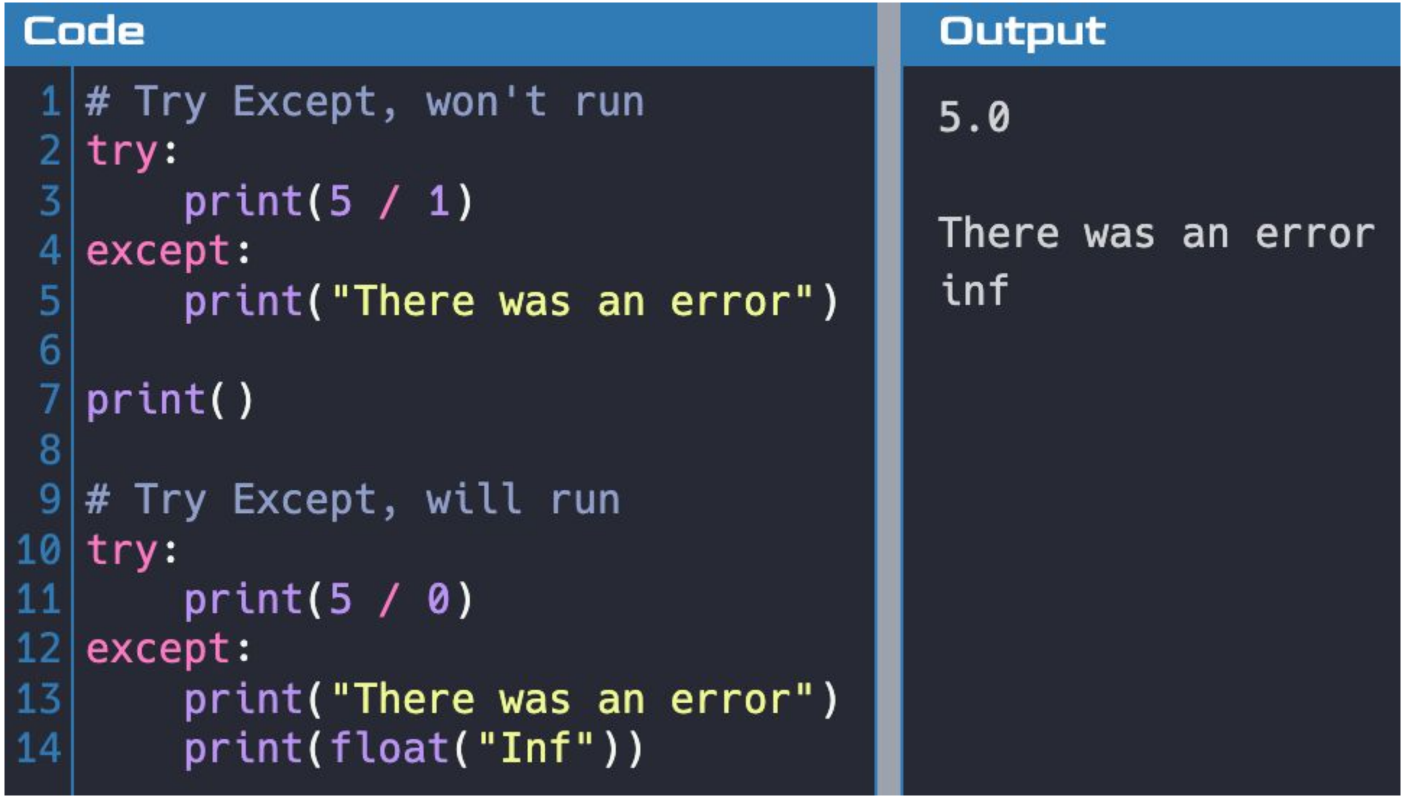
Output

```
Line 2: ZeroDivisionError: integer division or modulo by zero
```

Try Except

- Exceptions thrown inside a try block won’t end the program, just skip the rest of the block
- When an exception is thrown inside a try block, the except block runs
    - Except (“catch”) allows you to handle errors
- Can output a message, fix the problem, log it
- Can have multiple except statements which catch different exceptions

Try Except Code



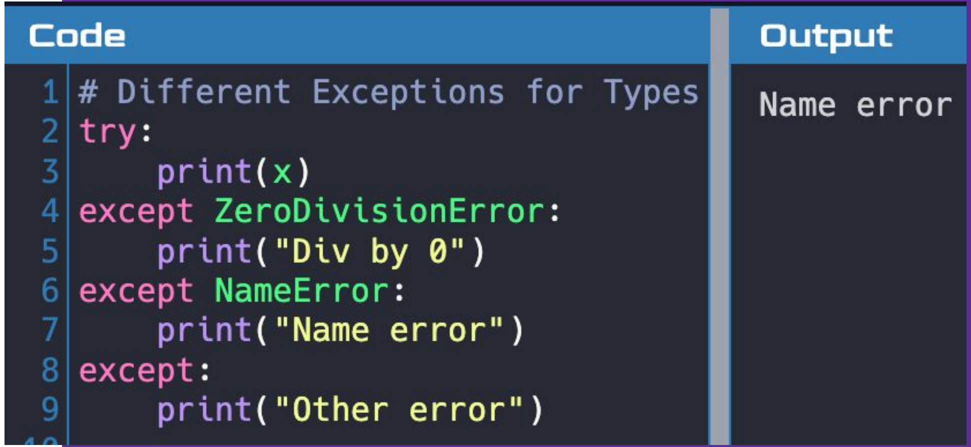
Code

```python
# Try Except, won't run
try:
    print(5 / 1)
except:
    print("There was an error")

print()

# Try Except, will run
try:
    print(5 / 0)
except:
    print("There was an error")
    print(float("Inf"))
```

Output

```
5.0

There was an error
inf
```

Multiple Except Statements

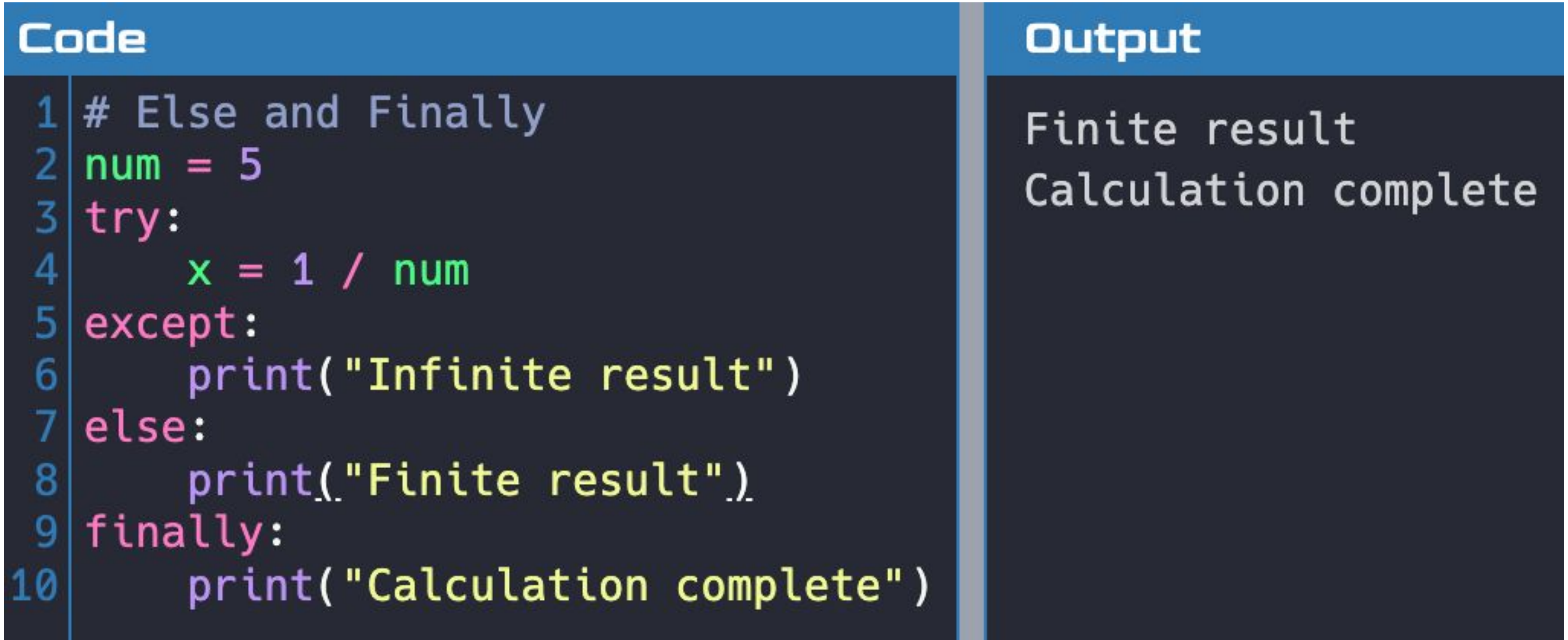


```python
# Different Exceptions for Types
try:
    print(x)
except ZeroDivisionError:
    print("Div by 0")
except NameError:
    print("Name error")
except:
    print("Other error")
```

```
Name error
```

Else and Finally

- Else runs if there’s no error thrown
- Finally runs regardless of whether or not an error is raised



Code

```python
# Else and Finally
num = 5
try:
    x = 1 / num
except:
    print("Infinite result")
else:
    print("Finite result")
finally:
    print("Calculation complete")
```

Output

```
Finite result
Calculation complete
```

Practice Problems

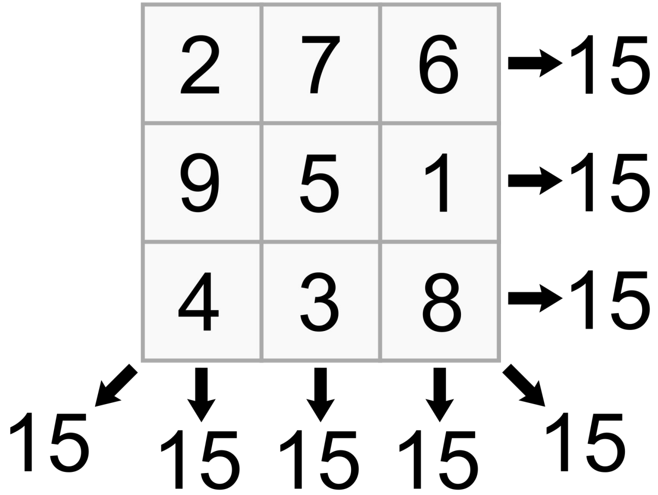
Practice Problem 1:

Splotvian Cuisine

- Src: HSPC 2019 Problems, #4
- Goal: Write a Python program that will take in a list of dish names and update them to be Splotvian. To do this, add " and chips" to the name of any dish that does not start with a q. Allow the user to input until the word "DONE" is inputted, then output the new menu.
- Relevant Information:
  - Ex:
    - Input:
      - pie
      - biryani
      - quinoa
      - steak tartare
    - Output:
      - pie and chips
      - biryani and chips
      - quinoa
      - steak tartare and chips

Practice Problem 2: Magic Tic Tac Toe

- Src: HSPC Problems, #10
- Goal: Write a Python program that uses a magic square to represent a tic tac toe board. Take 3 inputs for X and 3 for O and determine if X or O has already won. If not, determine if X has a winning move. If not, determine if X has a blocking move (can stop the other player from winning).
- Relevant Information:
  - Ex 1:
    - Input:
      - Enter first X move: 6, Enter first O move: 9
      - Enter second X move: 1, Enter second O move: 7
      - Enter third X move: 5, Enter third O move: 8
    - Output: Play 4 to win
  - Ex 2:
    - Input:
      - Enter first X move: 6, Enter first O move: 1
      - Enter second X move: 5, Enter second O move: 8
      - Enter third X move: 4, Enter third O move: 3
    - Output: X has already won
  - Final possible output: Play [num] to block



Practice Problem 3:

Overlap

- Src: HSPC 2019 Problems, #6
- Goal: Write a Python program that will determine the overlap between 2 pairs of numbers.
- Relevant Information:
    - You can assume all inputs are integers
    - Ex 1:
        - Input:
            - Enter interval 1: 2 3
            - Enter interval 2: 4 5
        - Output: There is no overlap between the two intervals
    - Ex 2:
        - Input:
            - Enter interval 1: 2 6
            - Enter interval 2: 4 5
        - Output: There is an overlap of 1 units between the two intervals

Next Meeting?