2.0 Python

Advanced

Lee Sing Jie

Outcome

- You will learn advance data structures
- You will learn how to write cleaner code, and easy to read
- You will be dealing with complex logic branching
- You will be building a virtual Tic Tac Toe game

Python Recap

- Hello World
- if / else
- input / output
- loops

Python Recap

Hello World

```
print("Hello World")
```

Python Recap

if / else

```
is_world_round = False
if is_world_round:
    print("Hello World")
else:
    print("Flat World")
```

Python Recap

input / output

```
value_1 = input("Please enter value 1: ")

print('The value is', value_1)
```

Python Recap

Loop

```
while True:
  print("Hello World")

while i < 10:
  print("Hello World")
  i = i + 1
```

Loops

For Loop

```python
print("Hi")
for i in range(3):
    print("Hello World")
print("Bye")
```

Loops

For Loop

```
print("Hi")
for i in range(3):
    print("Hello World")
print("Bye")
```

```
Hi
Hello World
Hello World
Hello World
Bye
```

Loops

For Loop

```
print("Hi")
for i in range(3):
    print("Hello World", i)
print("Bye")
```

```
Hi
Hello World, 0
Hello World, 1
Hello World, 2
Bye
```

Loops

For Loop

```
print("Hi")
for i in range(1,3):
    print("Hello World", i)
print("Bye")
```

```
Hi
Hello World, 1
Hello World, 2
Bye
```

Loops

For Loop vs While Loop

```
print("Hi")
for i in range(3):
    print("Hello World")
print("Bye")
```

```
print("Hi")
i = 0
while i < 3
    print("Hello World")
    i = i + 1
print("Bye")
```

Loops

For Loop vs While Loop

```
print("Hi")
for i in range(3):
    print("Hello World")
print("Bye")
```

```
print("Hi")
i = 0
while i < 3
    print("Hello World")
    i = i + 1
print("Bye")
```

Achieves the same purpose

Loops

For Loop vs While Loop

```
print("Hi")
for i in range(3):
    print("Hello World")
print("Bye")
```

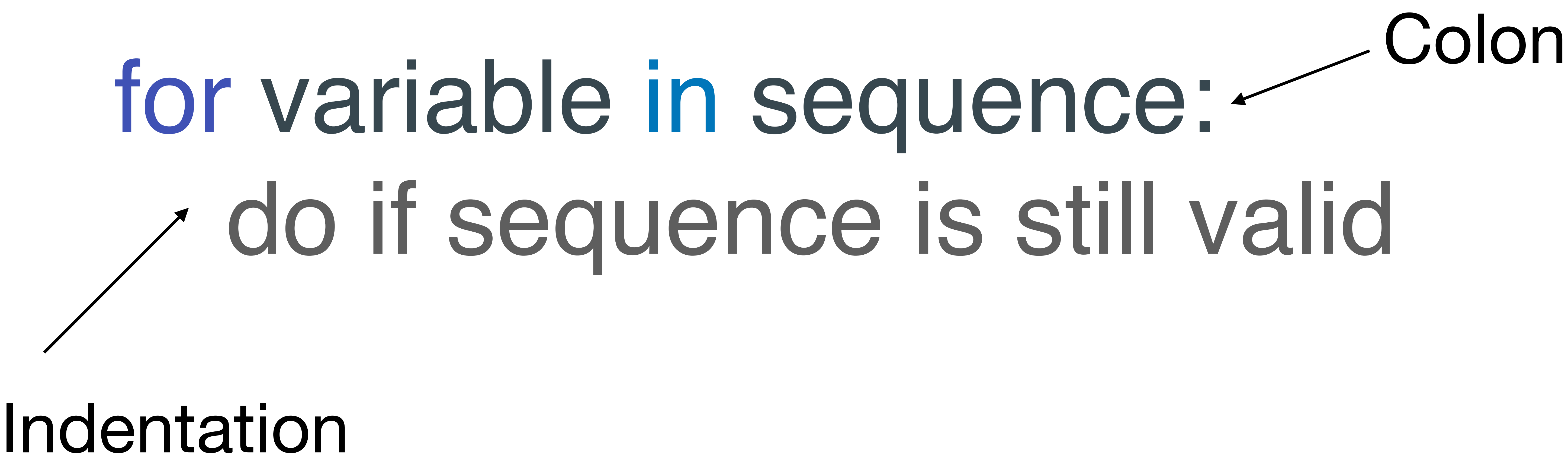
But way neater

```
print("Hi")
i = 0
while i < 3
    print("Hello World")
    i = i + 1
print("Bye")
```

Achieves the same purpose

For Loop

For Loop statement



```
for variable in sequence:
    do if sequence is still valid
```

Colon

Indentation

Loops

For Loop

```
for i in range(5):
    print(i)

for i in "Singapore":
    print(i)
```

Loops

For Loop

```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

```
for i in "Singapore":
    print(i)
```

```
S
i
n
g
a
p
o
r
e
```

Lab

Word has ‘a’

Lab

Word has ‘a’

```
Please type a word: singapore

Your word has ‘a’ character


Please type a word: hello

Your word has no ‘a’ character
```

Data Structures

Different Data Types

- String / Int
- List
- Dict
- Set

Data Structures

Int / String

```
value_1 = "Singapore Chinese Girls School"  string
value_2 = 123                               int
value_3 = "123"                             string
value_4 = "123.4"                           string
```

Data Structures

String

```
value = "Hello World”

print(value) # Hello World
print(value[1]) # e
print(value[1:]) # ello World
print(value[1:3]) # el
print(value[:1]) # H
```

Data Structures

String - Index accessor

```
value = "Hello World”

print(value) # Hello World
print(value[1]) # e
print(value[0]) # H
print(value[4]) # o
```

Data Structures

String - Start from

```
value = "Hello World”

print(value) # Hello World
print(value[1:]) # ello World
print(value[0:]) # Hello World
print(value[6:]) # World
```

Data Structures

String - Start from and stop at

```
value = "Hello World”

print(value) # Hello World
print(value[1:2]) # e
print(value[2:7]) # llo W
print(value[6:1]) #
print(value[6:7]) # W
```

Data Structures

String - Stop at

```
value = "Hello World”

print(value) # Hello World
print(value[:2]) # He
print(value[:7]) # Hello W
print(value[:1]) # H
```

Data Structures

String - ‘+’

```
value = "Hello World”

print(value[0] + value[4] + value[7]) # Hoo
print(value[0] + value[4] + value[7] + "yaa") # Hooyaa
```

Data Structures

String - ‘+’

```
value = "Hello World”

print(len(value)) # 11
print(len("Singapore")) # 9
print(len("A")) # 1
print(len(" ")) # 1
```

Data Structures

String - in

```
value = "Hello World”

print('e' in value) # True
print('b' in "banana") # True
print('c' in "banana") # False
print('ana' in "banana") # True
```

Lab

Word has ‘an’

Lab

Word has ‘an’

```
Please type a word: banana

Your word has a ‘an’ string


Please type a word: helloa

Your word has no ‘an’ string
```

Data structures

List

```
numbers = 14

fruits = 'orange'
```

Data structures

List

```
numbers = [14, 15, 18, 0, 1]

fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
```

Data structures

List - Index Accessor

```
fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
print(fruits[1]) # apple
```

Data structures

List - Start from

```
fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
print(fruits[1:]) # [‘apple’, ‘pear’, ‘banana’, ‘kiwi’, ‘apple’ banana’]
print(fruits[4:]) # [‘kiwi’, ‘apple’ banana’]
```

Data structures

List - Stop at

```
fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
print(fruits[:1]) # [‘orange’]
print(fruits[:4]) # [‘orange’, ‘apple’, ‘pear’, ‘banana’]
```

Data structures

List - Start from and Stop at

```
fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
print(fruits[3:4]) # [‘banana’]
print(fruits[0:2]) # [‘orange’, ‘apple’]
```

Data structures

List - len

```
fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
print(len(fruits)) # 7
```

Data structures

List - ‘+’

```
fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
fruits = fruits + [‘grapes']
print(len(fruits)) # 8
```

Data structures

List

```
numbers = [14, 15, 18, 0, 1]
print(len(numbers)) # 5
```

Data structures

List

```
numbers = [14, 15, 18, 0, 1]
numbers = numbers + [0]
print(len(numbers)) # 6
```

Data structures

List

```
numbers = [14, 15, 18, 0, 1]
print(numbers[1:4]) # [15, 18, 0]
```

Data structures

List - count

```
fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple',
'banana']
print(fruits.count('pear')) # 1
print(fruits.count('banana')) # 2
print(fruits.count('durian')) # 0
```

Lab

Shopping Cart

Lab

Shopping Cart

```
Add items into shopping cart: banana
Do you want to keep adding? (Y/N): Y

Add items into shopping cart: apple
Do you want to keep adding? (Y/N): N

You have a total of 2 items in your cart.
Your first item is banana.
Your last item is apple.
```

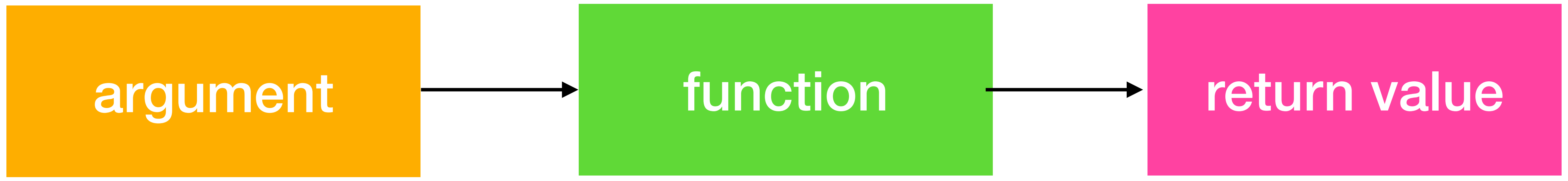
Data structures

Function

```
result = add(2, 1)
salad = makeSalad(lettuce, nuts, dressing)
coffee = makeCoffee(power, milk)
```

Data structures

Function



argument

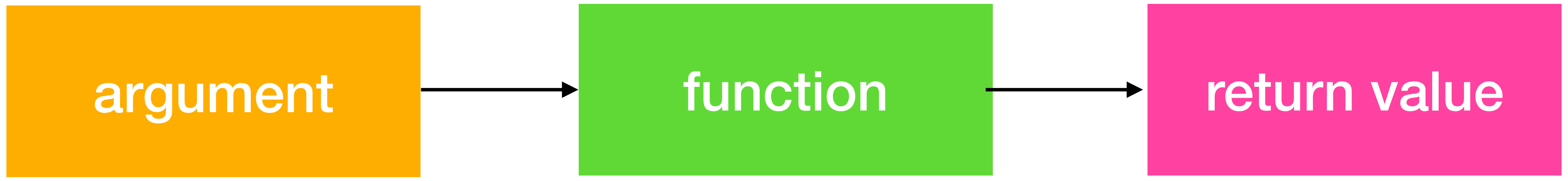
function

return value

```
result = add(2, 1)
salad = makeSalad(lettuce, nuts, dressing)
coffee = makeCoffee(power, milk)
```

Data structures

Function



```
result = add(2, 1)
salad = makeSalad(lettuce, nuts, dressing)
coffee = makeCoffee(power, milk)
```

Data structures

Function

```
value = abs(-5)
print(value) # 5

value = abs(10)
print(value) # 10
```

Data structures

Function

```
value = abs(-5)
print(value) # 5

another_value = value + abs(-8)
print(another_value) # 13

yet_another_value = abs(value + another_value + -10)
print(yet_another_value) # 8
```

Data structures

Function

```
def add(value1, value2):
    return value1 + value2
```

Data structures

Function

```
def add(value1, value2):
    return value1 * 2 + value2 * 2
```

Data structures

Function

```
def add(value1, value2):
    return value1 * 2 + value2 *2

add(1,1) # 4
add(2,3) # 10
add(-1, 5) # 8
```

Data structures

Function

```
def helloWorld():
    print("Hello World!")

helloWorld()
helloWorld()
helloWorld()
```

Data structures

Function + Loop

```
def helloWorld():
    print("Hello World!")

for index in range(3):
    helloWorld()
```

Data structures

Function statement

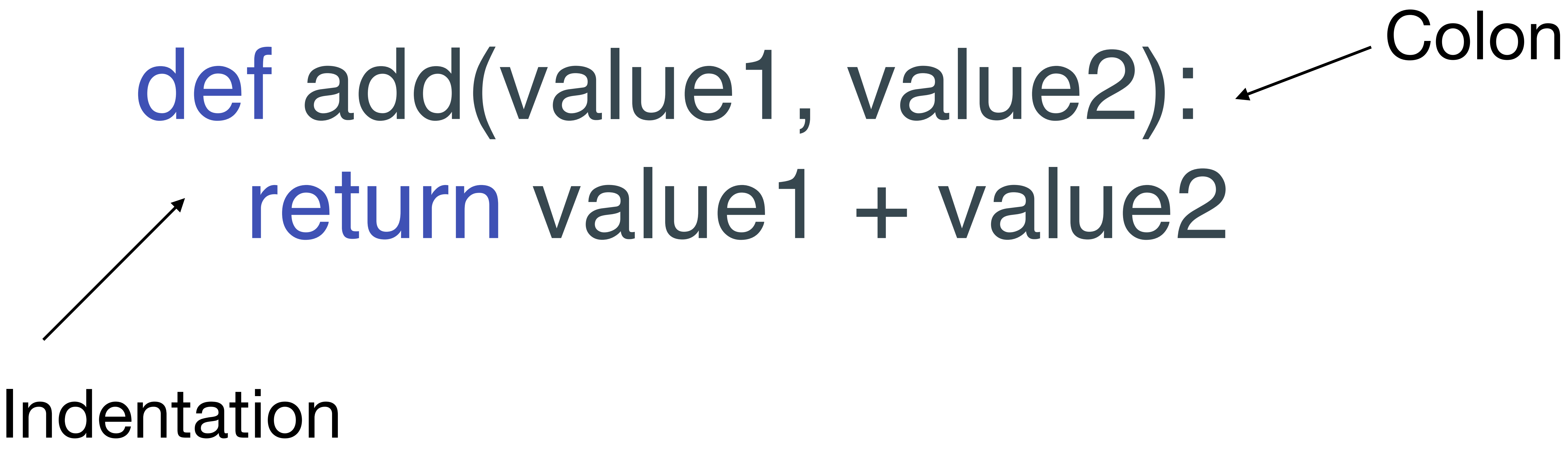
```
def function_name(arguments):
    do this process within function
    return some_value
```

Colon

Indentation

Data structures

Function statement



```
def add(value1, value2):
    return value1 + value2
```

Colon

Indentation

Data structures

Purpose of Functions

- Group related statements together
- Prevent repeated code
- Reflect mental model and abstraction to details
- Reusable code

Data structures

Purpose of Functions

```
print("Menu for Today")
print("1. Fish and Chips")
print("2. Fish Burger")
print("3. Chicken Chop")
print("4. Pork Chop")
print("5. Pepsi")
```

Data structures

Purpose of Functions

```
print("Menu for Today")
print("1. Fish and Chips")
print("2. Fish Burger")
print("3. Chicken Chop")
print("4. Pork Chop")
print("5. Pepsi")
```

```
def print_menu:
    print("Menu for Today")
    print("1. Fish and Chips")
    print("2. Fish Burger")
    print("3. Chicken Chop")
    print("4. Pork Chop")
    print("5. Pepsi”)

print_menu()
```

Data structures

Purpose of Functions

Find the area of a circle of a given radius: 2, 4, 10, 14, where pi = 3.142

```
pi = 3.142
```

```
print(pi * 2 * 2)
print(pi * 4 * 4)
print(pi * 10 * 10)
print(pi * 14 * 14)
```

Data structures

Purpose of Functions

Find the area of a circle of a given radius: 2, 4, 10, 14, where pi = 3.142

```
pi = 3.142

print(pi * 2 * 2)
print(pi * 4 * 4)
print(pi * 10 * 10)
print(pi * 14 * 14)
```

```
def area_of_circle(radius):
    return 3.142 * radius * radius

print(area_of_circle(2))
print(area_of_circle(4))
print(area_of_circle(10))
print(area_of_circle(14))
```

Data structures

Purpose of Functions

```
def area_of_circle(radius):
    return 3.142 * radius * radius

def print_and_calculate(radius):
    area = area_of_circle(radius)
    print(area)

print_and_calculate(2)
print_and_calculate(4)
print_and_calculate(10)
print_and_calculate(14)
```

Lab

Function - abs

Lab

Function - abs

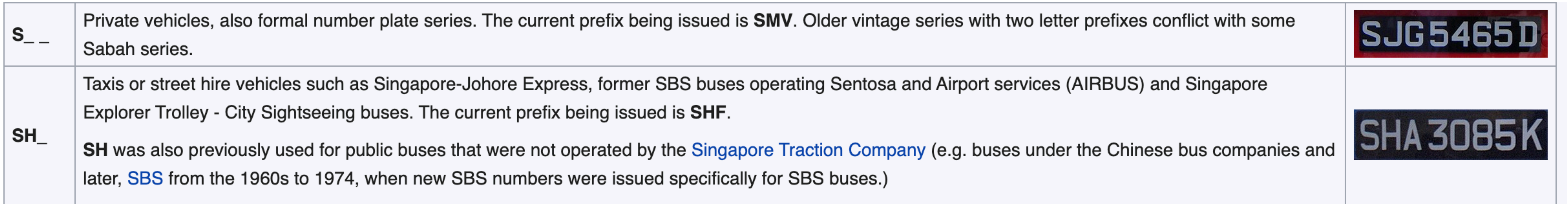
```
Enter value to be converted to absolute: -12

Absolute value of -12 is 12.
```

Lab

Function - Private Car / Taxi Plate

Lab

Function - Private Car / Taxi Plate



S_ _	Private vehicles, also formal number plate series. The current prefix being issued is **SMV**. Older vintage series with two letter prefixes conflict with some Sabah series.	SJG 5465 D
SH_	Taxis or street hire vehicles such as Singapore-Johore Express, former SBS buses operating Sentosa and Airport services (AIRBUS) and Singapore Explorer Trolley - City Sightseeing buses. The current prefix being issued is **SHF**. **SH** was also previously used for public buses that were not operated by the Singapore Traction Company (e.g. buses under the Chinese bus companies and later, SBS from the 1960s to 1974, when new SBS numbers were issued specifically for SBS buses.)	SHA 3085 K

From: Wikipedia

Lab

Function - Private Car / Taxi Plate

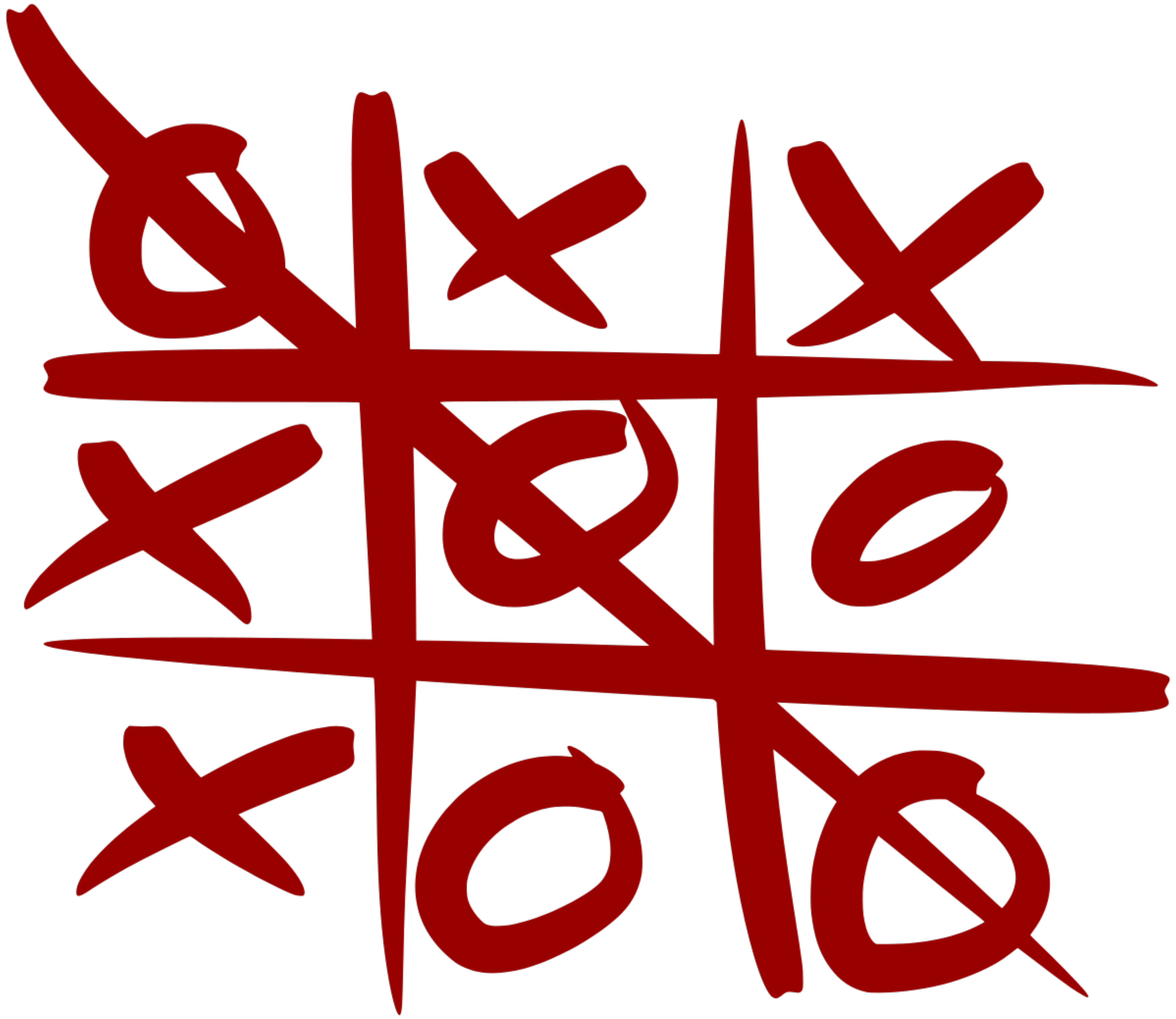
```
Enter License Plate: SHA9188L

This license plate belongs to a Taxi


Enter License Plate: SMH9188L

This license plate belongs to a Private Vehicle
```

Final Lab

Tic Tac Toe - Part 1

Final Lab

Function - Tic Tac Toe



Final Lab

Function - Tic Tac Toe

```
Player 1(X) - Input [0-8]:0

X |   |

  |   |

  |   |
```

Final Lab

Function - Tic Tac Toe

```
Player 2(O) - Input [0-8]:7

X | O | X

O | X | X

O | O |
```

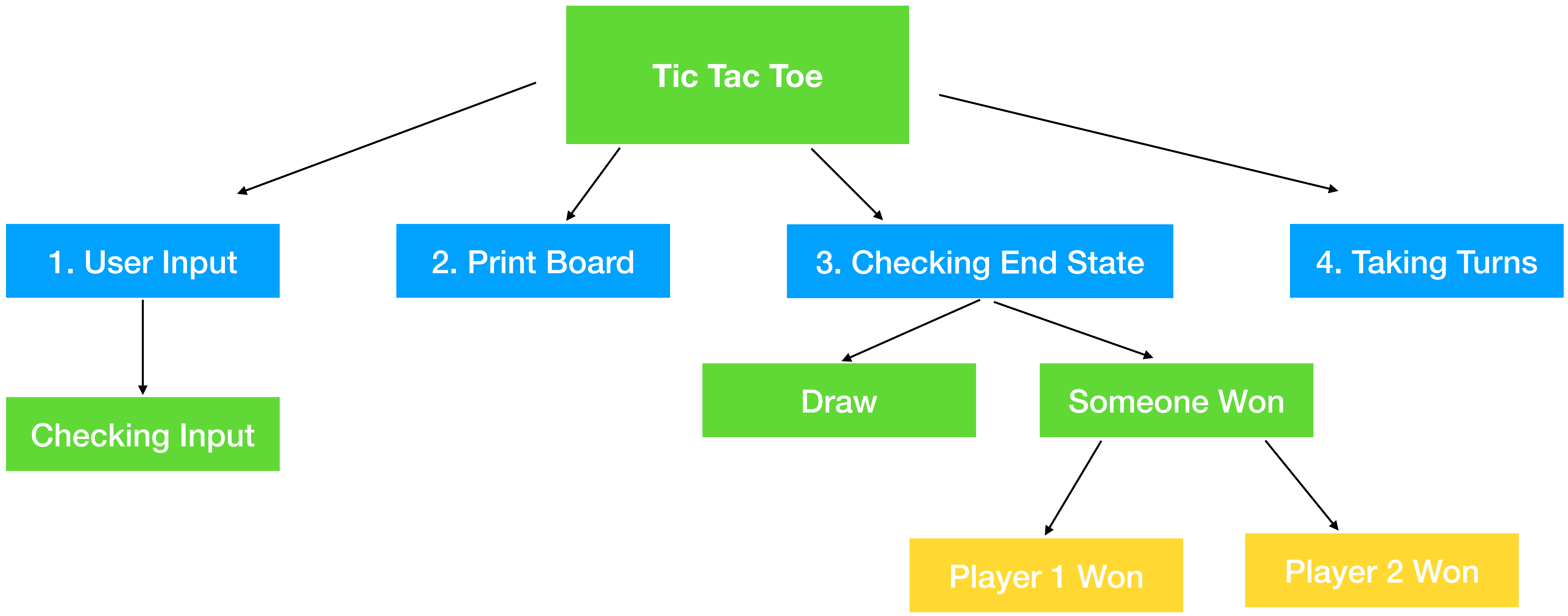
Final Lab

Function - Tic Tac Toe

```
Player 1(X) - Input [0-8]:8

X | O | X

O | X | X

O | O | X

Player 1(X) Won!
```

Final Lab

Function - Tic Tac Toe

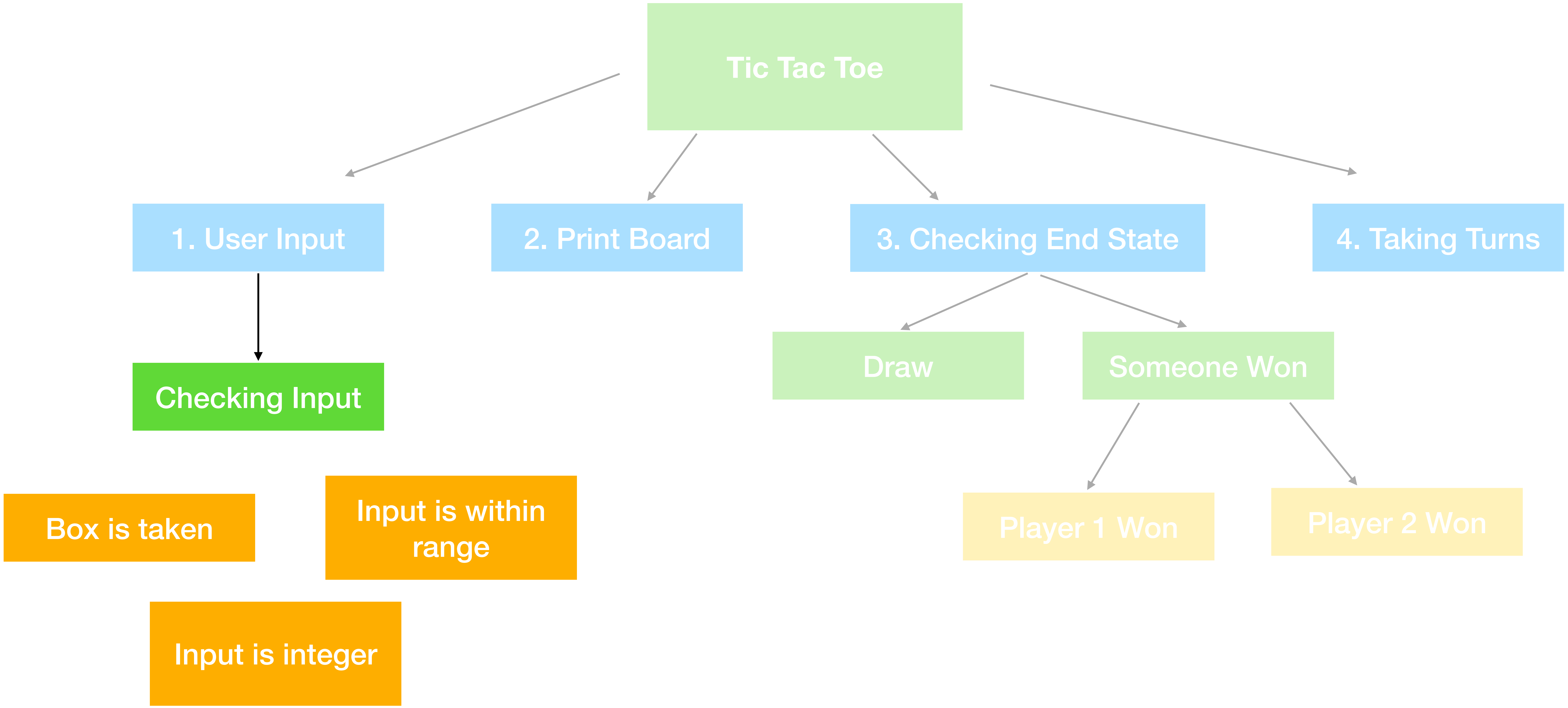


Final Lab

Tic Tac Toe - Part 2

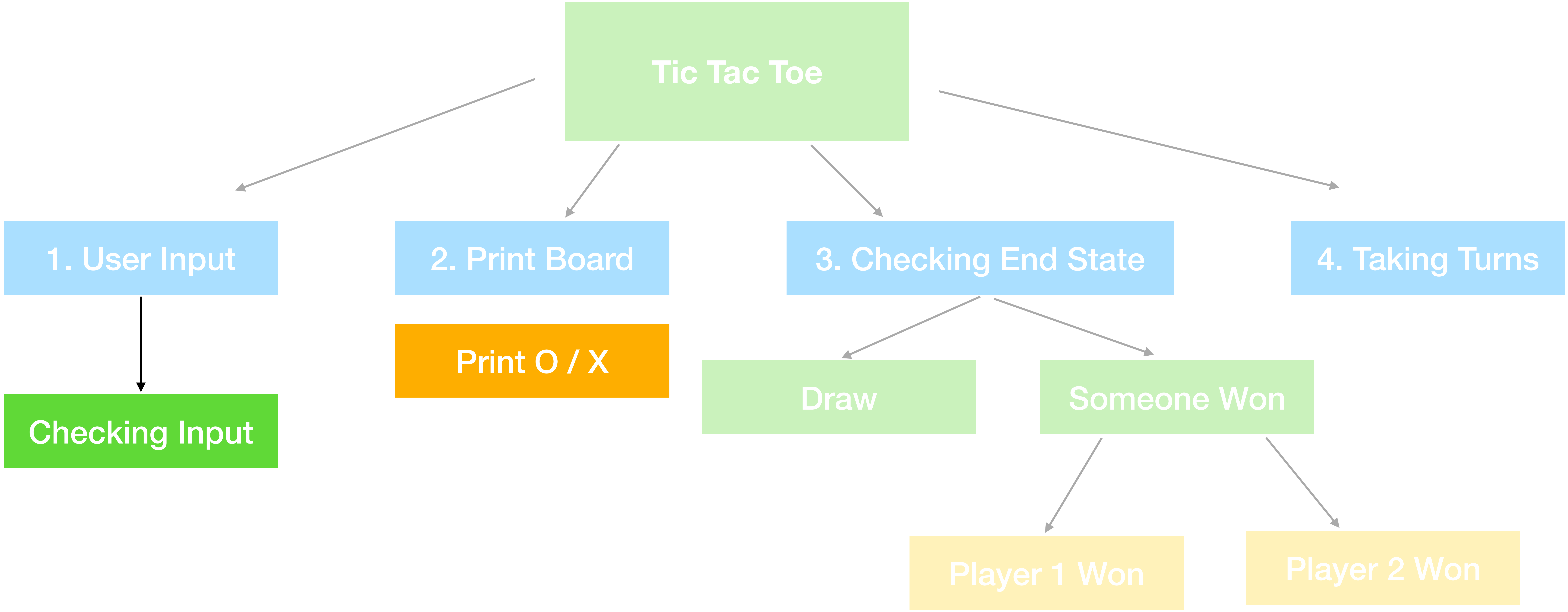
Final Lab

Function - Tic Tac Toe



Final Lab

Function - Tic Tac Toe

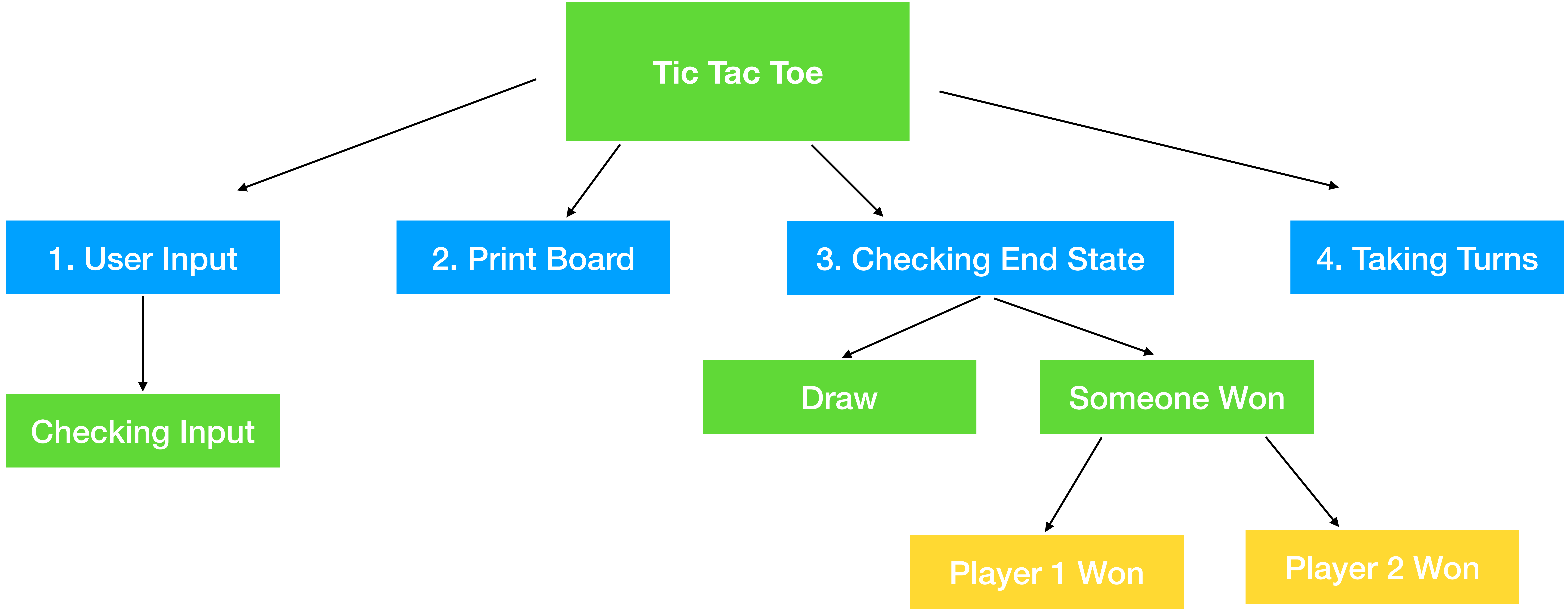


Final Lab

Tic Tac Toe - Part 3

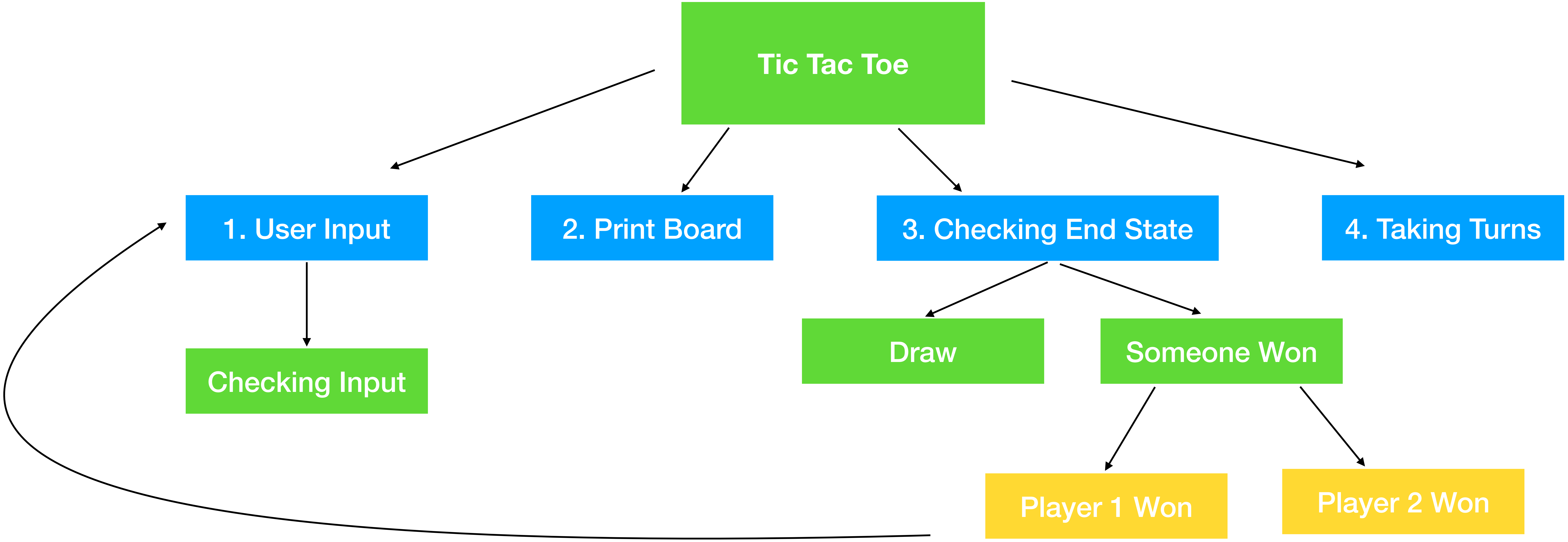
Final Lab

Function - Tic Tac Toe



Final Lab

Function - Tic Tac Toe



Tic Tac Toe

Looping

- Press ‘c’ to restart
- Other keys to quit

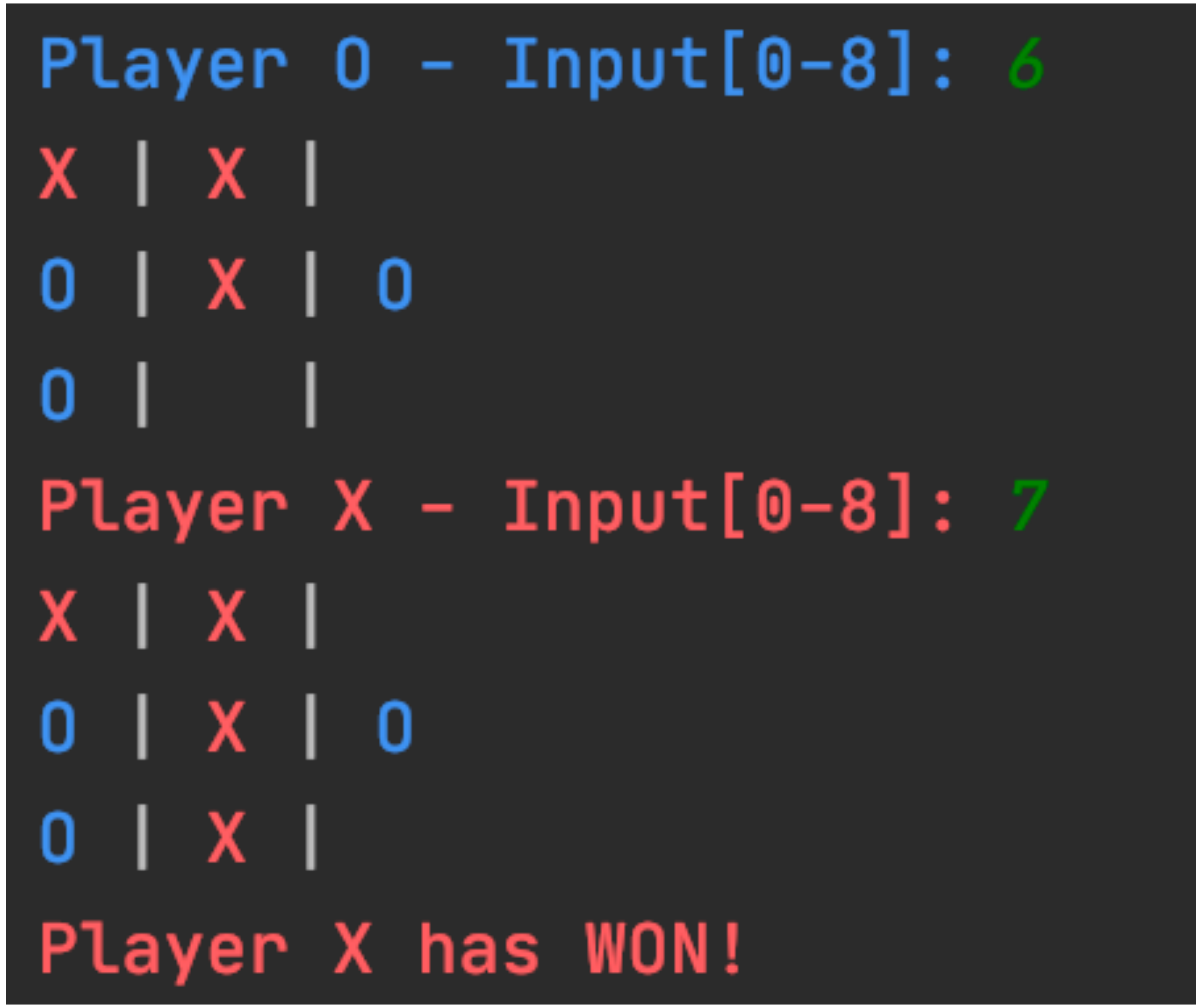
```
while True:
    v = input(“Press ‘c’ to restart; Other keys to quit")
    print(v)
```

Tic Tac Toe

Polishing - with Colours

Tic Tac Toe

Polishing - with Colours



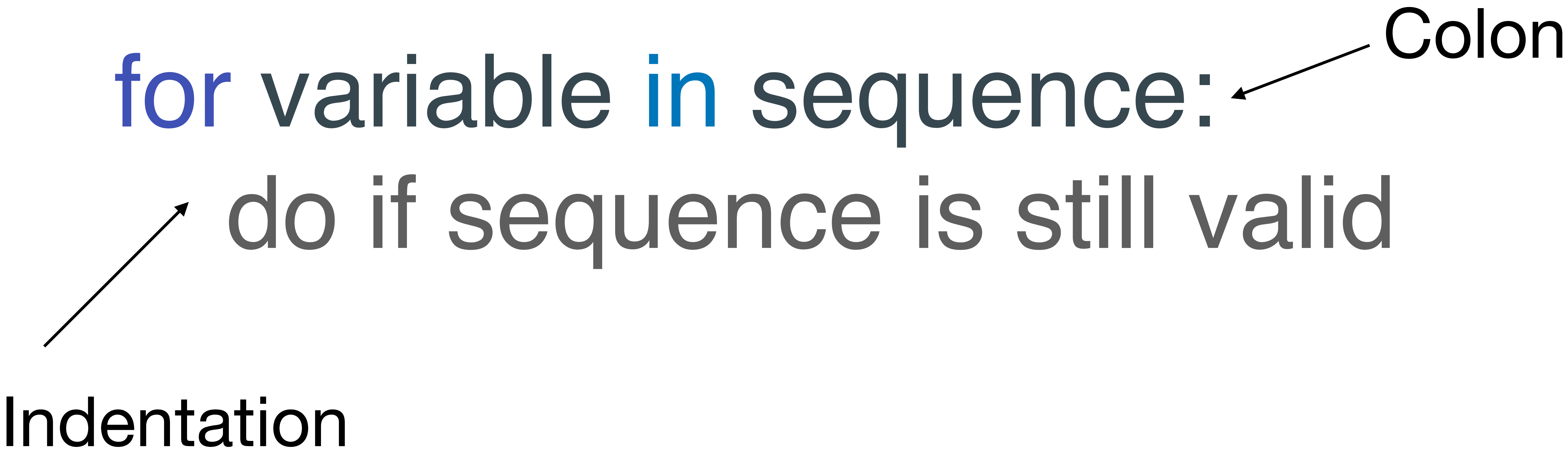
```
Player O - Input[0-8]: 6
X | X |
O | X | O
O |   |
Player X - Input[0-8]: 7
X | X |
O | X | O
O | X |
Player X has WON!
```

Recap

- For Loops
- Data Structures
  - Strings
  - List
  - Function
- Tic Tac Toe
  - Breaking into smaller problems
  - Loops
  - Colors

Recap

For Loops



```
for variable in sequence:
    do if sequence is still valid
```

Colon

Indentation

Recap

Data Structures - Strings

```
value = "Hello World”

print(value) # Hello World
print(value[1]) # e
print(value[2:7]) # llo W
print(value[:1]) # H
print(value[6:]) # World
```

Recap

Data Structures - List

```
fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
print(fruits[4]) # [‘kiwi’]
print(fruits[:1]) # [‘orange’]
print(fruits[1:4]) # [‘apple’, ‘pear’, ‘banana’]
print(fruits[4:]) # [‘kiwi’, ‘apple’, ‘banana’]
```

Recap

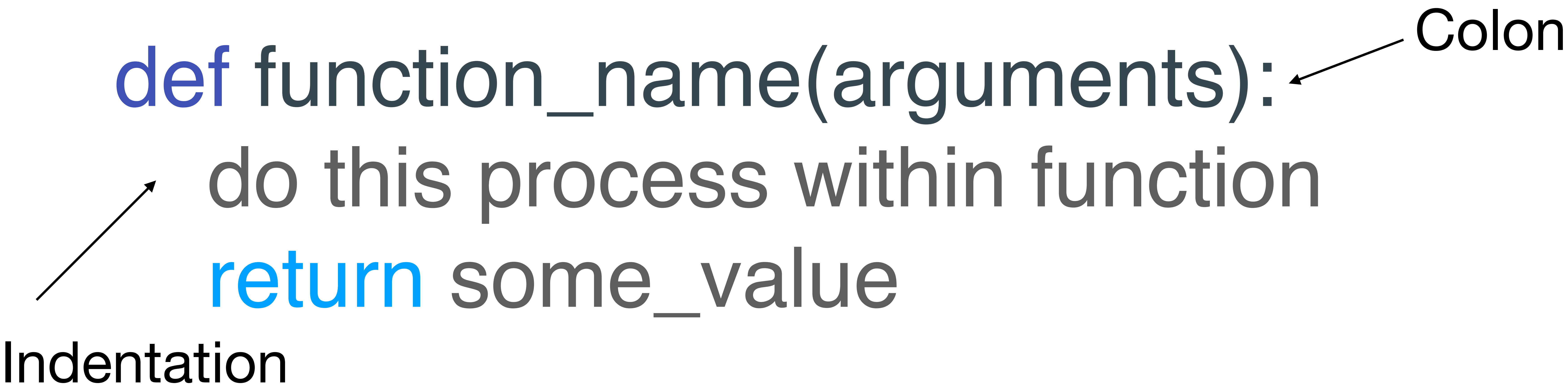
Data Structures - List

```
fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
print(len(fruits)) # 7
```

```
numbers = [14, 15, 18, 0, 1]
numbers = numbers + [0]
print(len(numbers)) # 6
```

Recap

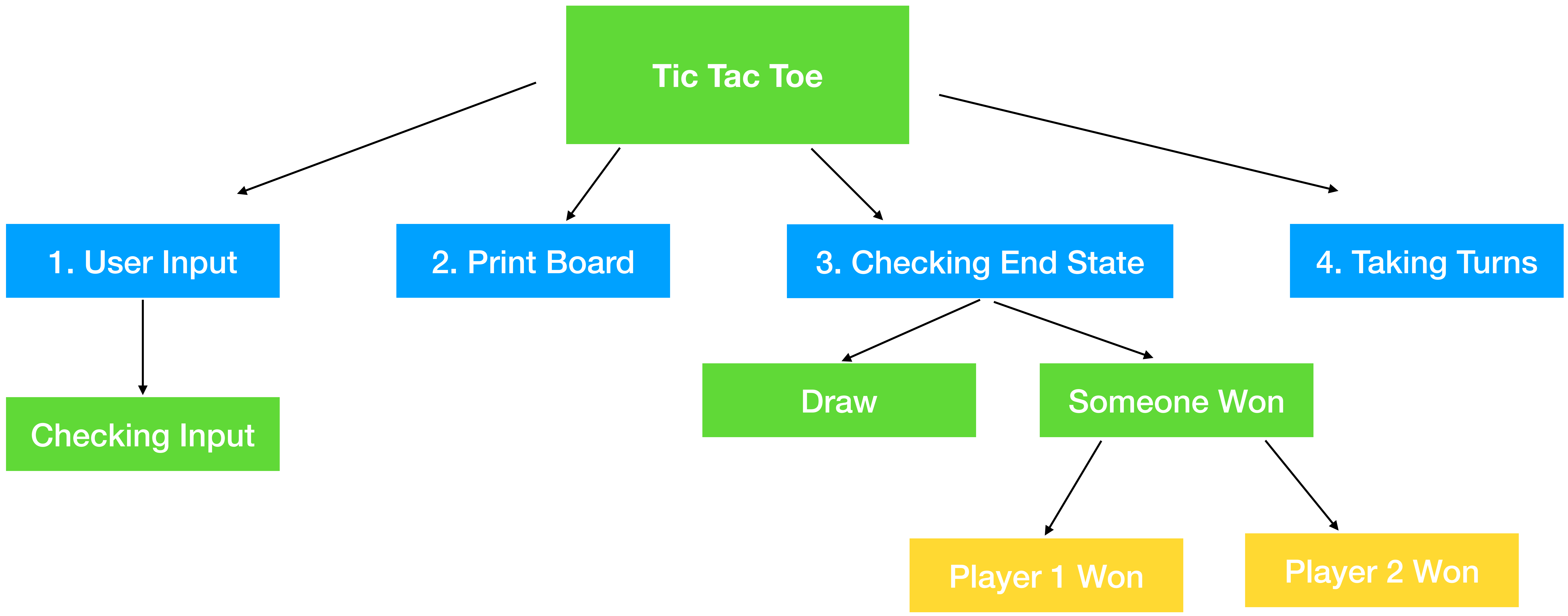
Data Structures - Function



```
def function_name(arguments):
    do this process within function
    return some_value
```

Final Lab

Function - Tic Tac Toe



Tic Tac Toe

1. User Input

Checking Input

2. Print Board

3. Checking End State

Draw

Someone Won

Player 1 Won

Player 2 Won

4. Taking Turns

Recap

Tic Tac Toe - Part 1

- Breaking into smaller problems
- Input / Output
- Loops
    - For Loops
    - While Loops
- Lists
- Functions

Recap

Tic Tac Toe - Part 1

```
Player 1 - Input [0-8]:8

0 | 0 | 0

0 | 0 | 0

0 | 0 | 1

Player 1 - Input [0-8]:2
```

Recap

Tic Tac Toe - Part 2

- Handling invalid cases of inputs
- Cleaning up the board and make game realistic

Recap

Tic Tac Toe - Part 2

```
Player X - Input [0-8]:blah

The input you entered is invalid

Player X - Input [0-8]:2
```

Recap

Tic Tac Toe - Part 2

```
Player X - Input [0-8]:8

O |   |

X |   |

  |   | X

Player O - Input [0-8]:2
```

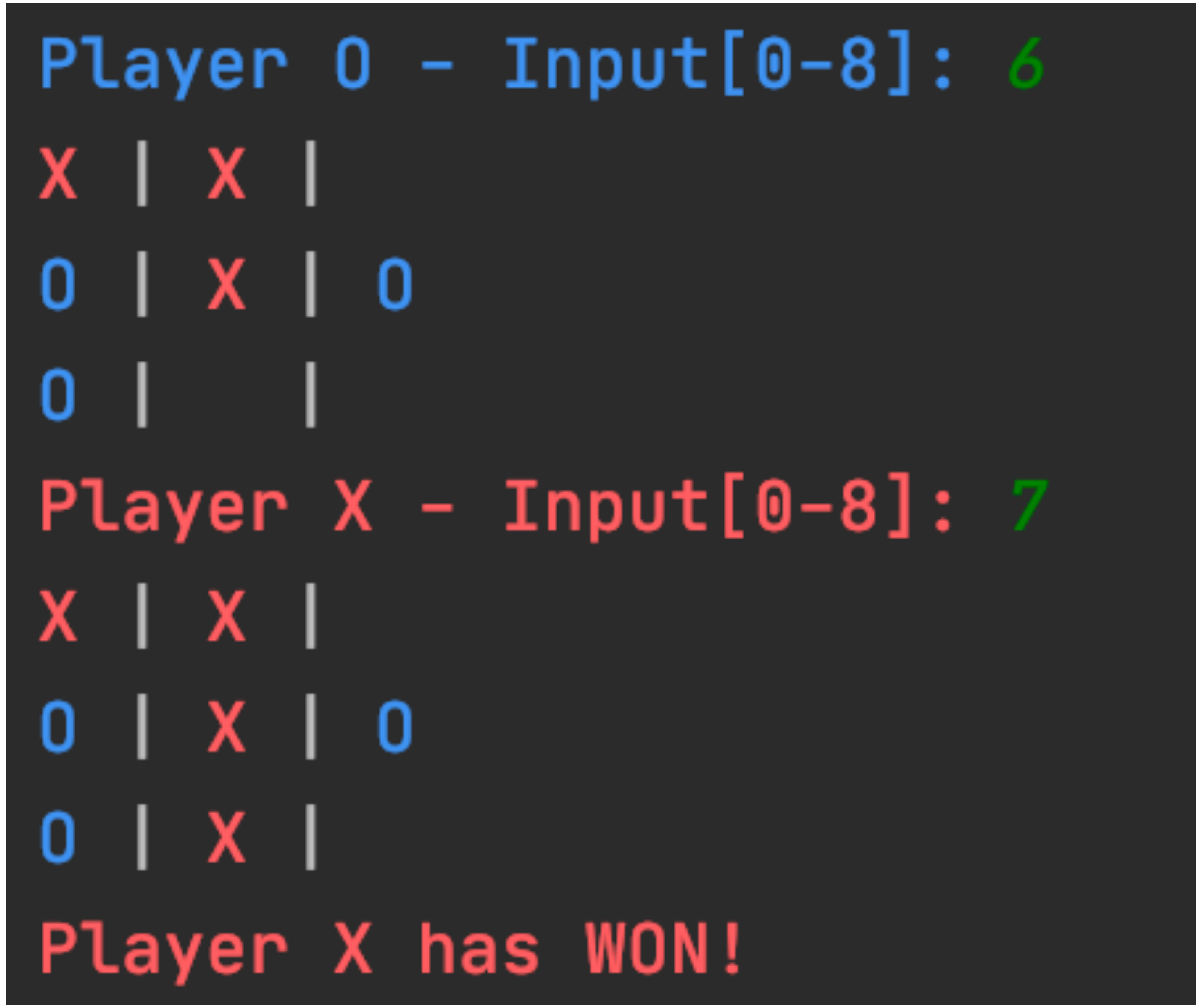
Recap

Tic Tac Toe - Part 3

- Beautifying the game with colours!
- Importing libraries

Recap

Tic Tac Toe - Part 3



```
Player O - Input[0-8]: 6
X | X |
O | X | O
O |   |
Player X - Input[0-8]: 7
X | X |
O | X | O
O | X |
Player X has WON!
```