# Conditionals

## CS 1111
## Introduction to Programming

## Spring 2019

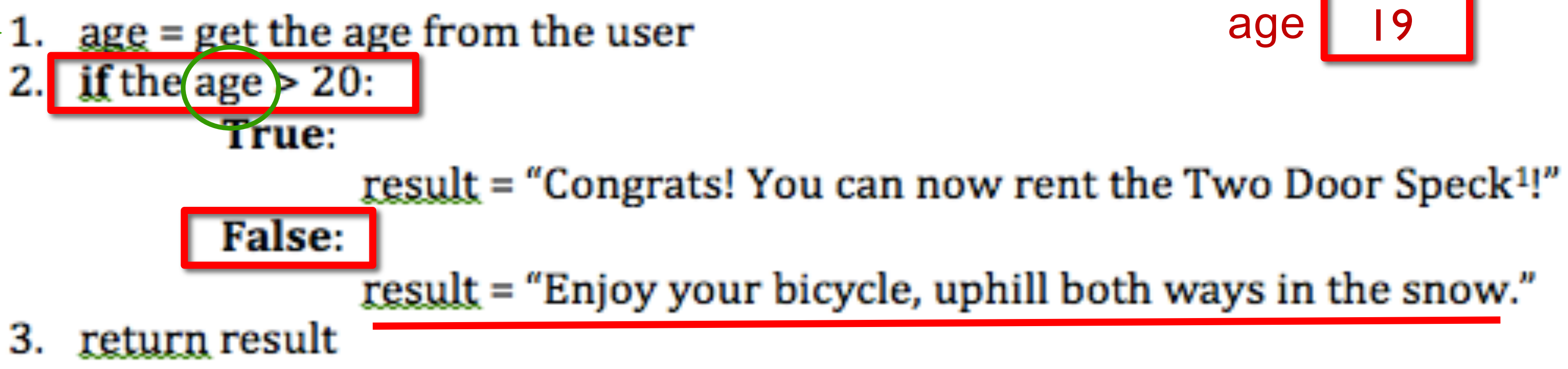
[*The Coder's Apprentice*, §6-6.2]
Based in part on "Agnostic Programming: Learning to Design and Test Basic Programming Algorithms"
by Kinga Dobolyi, Kindle]

# What is a Decision Statement?

- A statement that evaluates some conditions to true or false.

```
if condition:
     statement
     statement …
```

- A condition (or expression)
  - Must always evaluate to true or false, i.e., "Boolean expression"



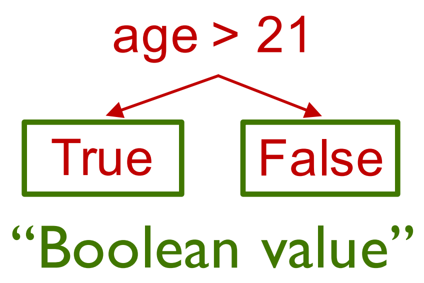
```
1.  age = get the age from the user
2.  if the age > 20:
          True:
                result = "Congrats! You can now rent the Two Door Speck¹!"
          False:
                result = "Enjoy your bicycle, uphill both ways in the snow."
3.  return result
```

age $\boxed{19}$

# Calculations that Evaluate to Boolean Values

- **< ≤ > ≥** all evaluate to true or false

  3 < 2 is False



- **==, !=** also evaluate to True or False

  3 == 3 is True

  3 == 4 is False

  "jello" != "blue" is True

  5 == 5.0 is True

  type(5) == type(5.0) is False

  '5' == '5.0' is False

  type('5') == type('5.0') is True

# Decision Structure

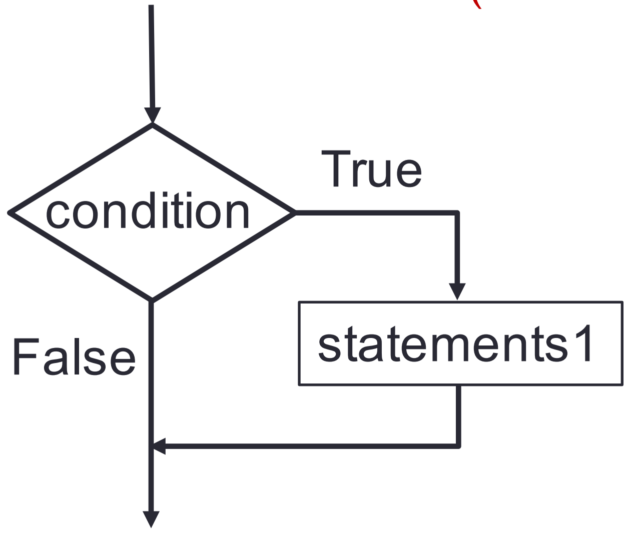
## Simple structure

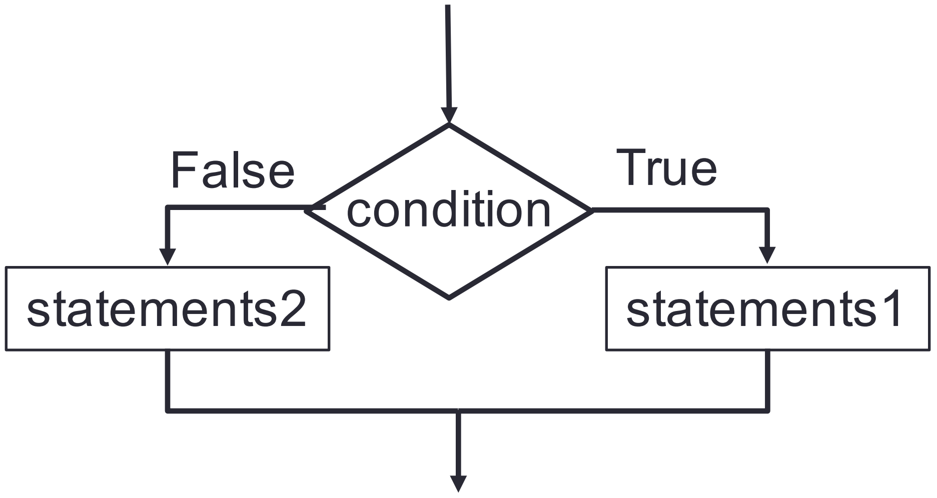
```
if condition:
    statements1
```

## Dual Structure

```
if condition:
    statements1
else:
    statements2
```

Indent the block of statements (sometimes called "body")





# Two Types of Decisions

## Sequence Decision

```
if condition:
    block of statements
    ...
if condition:
    block of statements
    ...
if condition:
    block of statements
    ...
else:
    block of statements
    ...
```

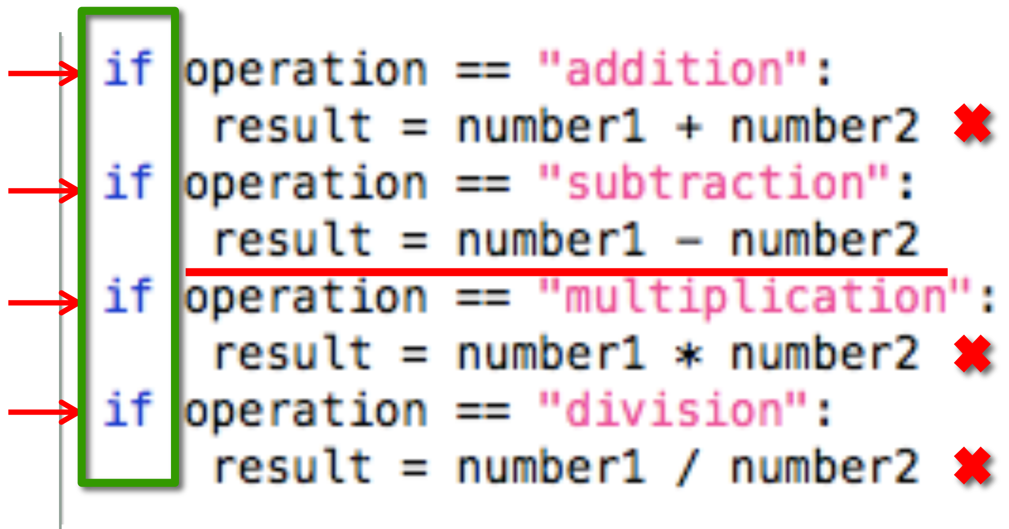
## Nested Decision

```
if condition:
    block of statements
    ...
elif:
    block of statements
    ...
elif:
    block of statements
    ...
else:
    block of statements
    ...
```

- Nested decisions remember the results of decisions made before them (in the same nesting)
- Independent decisions do not

# if Statements in Python

```
if operation is addition:
        True: result = number1 + number2
        False: do nothing
if operation is subtraction:
        True: result = number1 - number2
        False: do nothing
if operation is multiplication:
        True: result = number1 x number2
        False: do nothing
if operation is division:
        True: result = number1 / number2
        False: x = 3
```



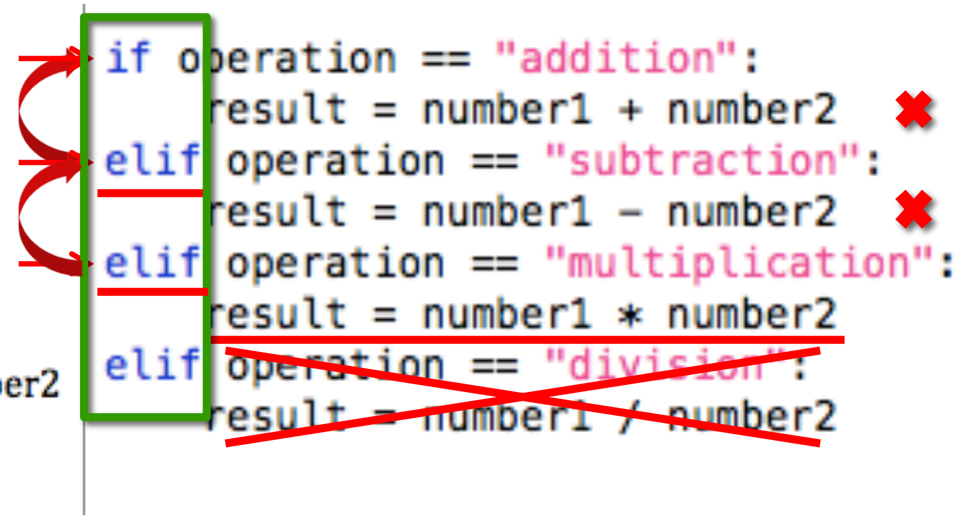
```python
if operation == "addition":
    result = number1 + number2   ✖
if operation == "subtraction":
    result = number1 - number2
if operation == "multiplication":
    result = number1 * number2   ✖
if operation == "division":
    result = number1 / number2   ✖
```

operation $\boxed{\text{“subtraction”}}$

- `if` is a keyword, the if statement must end in a colon
- What belongs to a particular if statement is indented

# `elif` Statements in Python

```
if operation is addition:
          True: result = number1 + number2
          False: if operation is subtraction:
                    True: result = number1 - number2
                    False: if operation is multiplication:
                              True: result = number1 x number2
                              False: if operation is division:
                                        True: result = number1 / number2
                                        False: x = 3
```

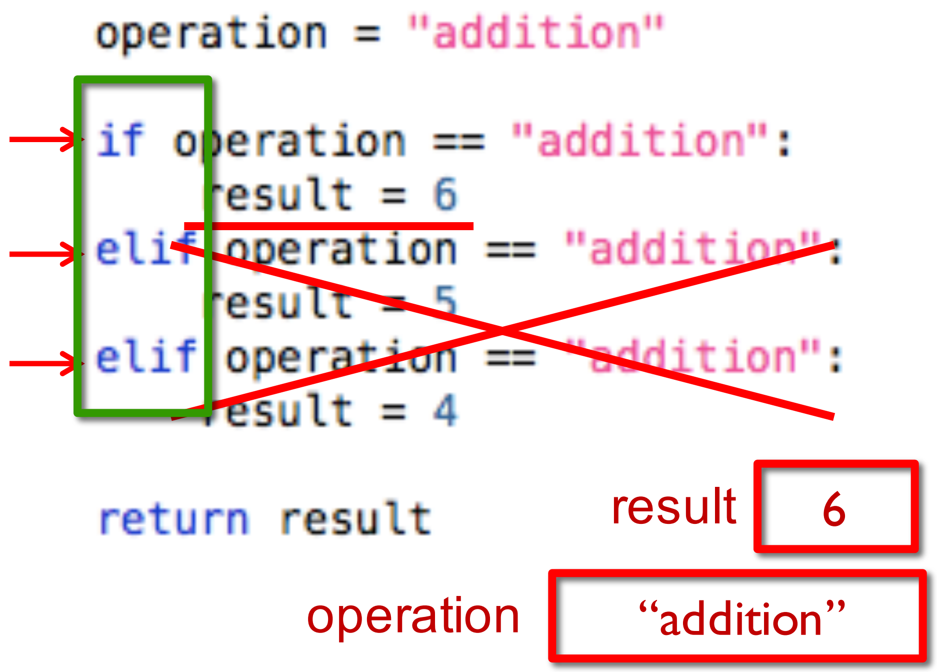


```
if operation == "addition":
    result = number1 + number2
elif operation == "subtraction":
    result = number1 - number2
elif operation == "multiplication":
    result = number1 * number2
~~elif operation == "division":~~
    ~~result = number1 / number2~~
```

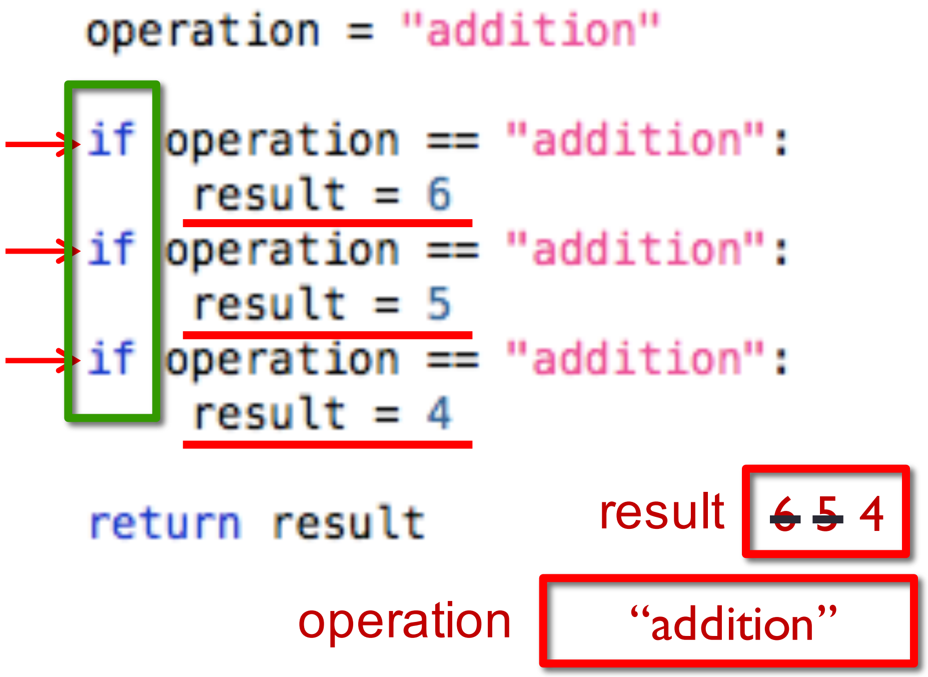
operation $\boxed{\text{“multiplication”}}$

- `elif` is a keyword; it stands for else if
- `elif` is attached to an `if` or `elif` before it, and indicates this `elif` is nested
- (you cannot have a standalone `elif`)

## if versus elif



```
operation = "addition"

if operation == "addition":
    result = 6
elif operation == "addition":
    result = 5
elif operation == "addition":
    result = 4

return result
```

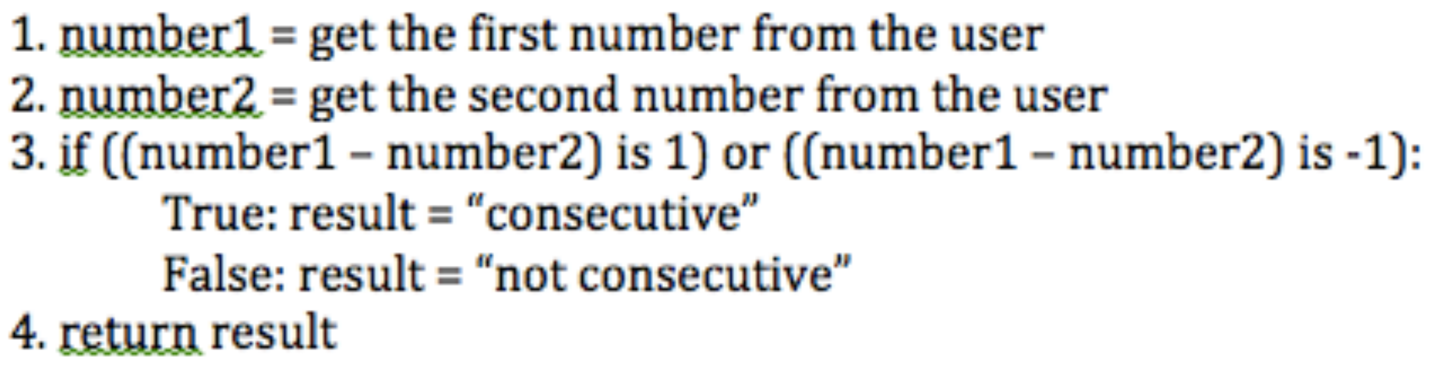


```
operation = "addition"

if operation == "addition":
    result = 6
if operation == "addition":
    result = 5
if operation == "addition":
    result = 4

return result
```

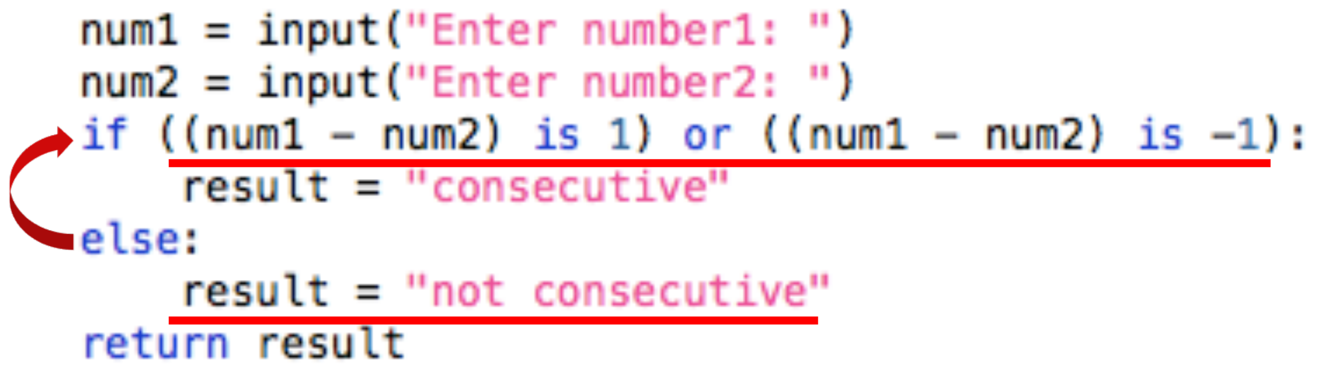
- The one on the left returns 6
- **if-elif** statements are nested, linked, and mutually exclusive.

- The one on the right returns 4
- The plain **if** statements are not mutually exclusive, don’t know about each other, and thus all `if` statements get executed

# else statements



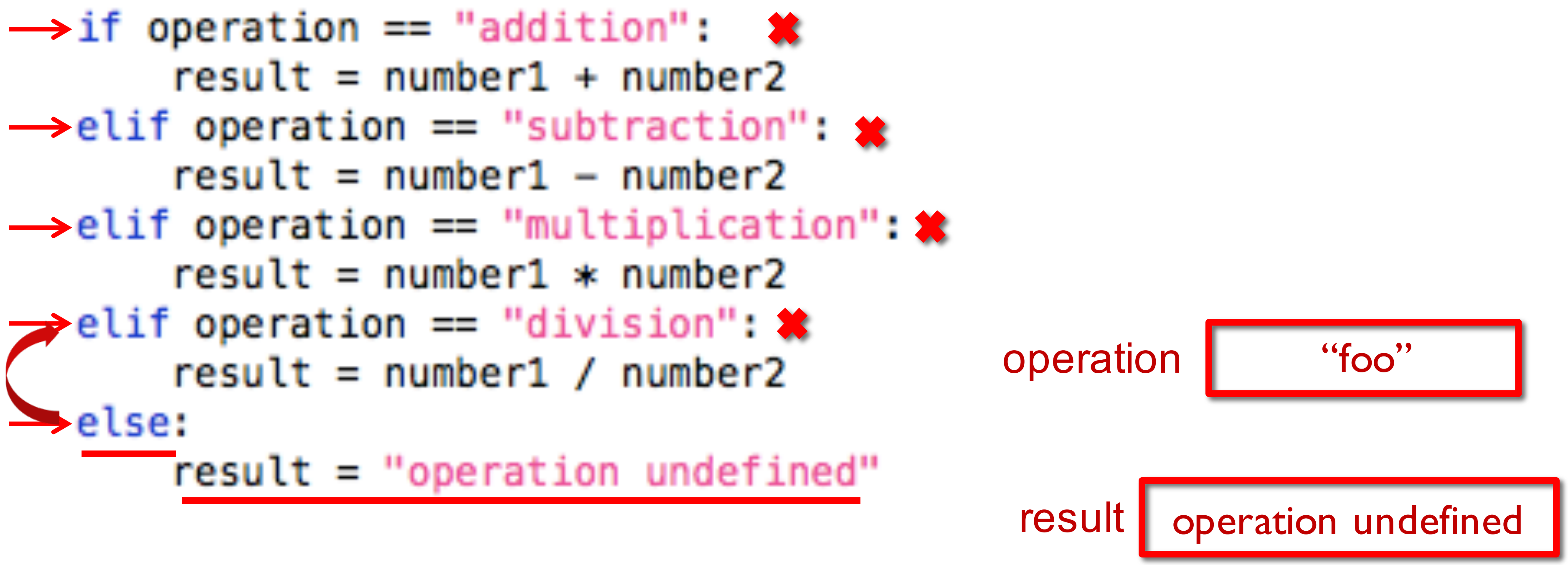
```
1. number1 = get the first number from the user
2. number2 = get the second number from the user
3. if ((number1 – number2) is 1) or ((number1 – number2) is -1):
        True: result = “consecutive”
        False: result = “not consecutive”
4. return result
```



```
num1 = input("Enter number1: ")
num2 = input("Enter number2: ")
if ((num1 - num2) is 1) or ((num1 - num2) is -1):
    result = "consecutive"
else:
    result = "not consecutive"
return result
```

- **`else`** is a keyword, linked to an `if` or `elif`, and get executed if the `if`/`elif` above it is false

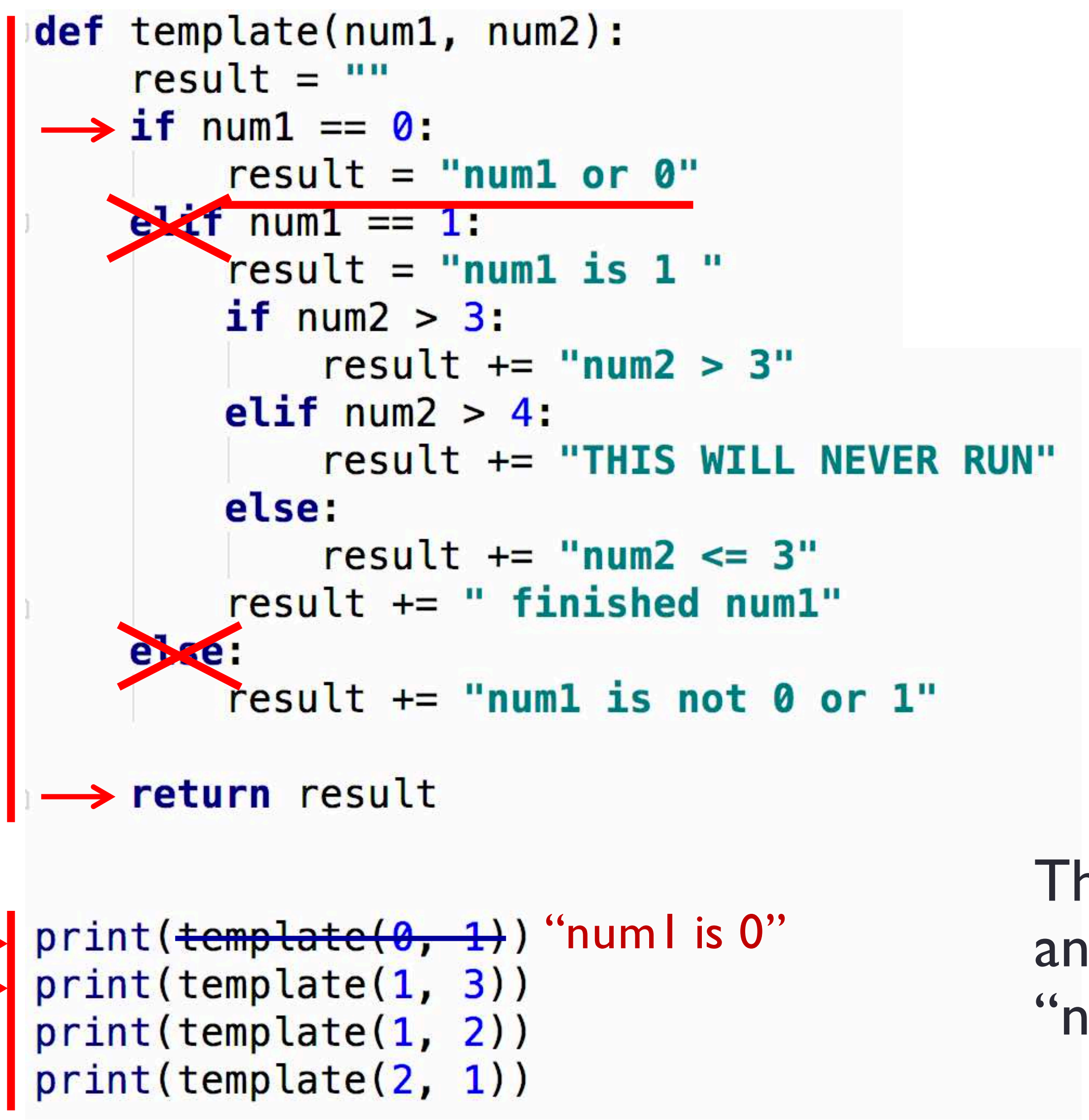
# else statements (2)



```
if operation == "addition":
    result = number1 + number2
elif operation == "subtraction":
    result = number1 - number2
elif operation == "multiplication":
    result = number1 * number2
elif operation == "division":
    result = number1 / number2
else:
    result = "operation undefined"
```

- `else` only gets executed if none of the `if` or `elif` before it are true

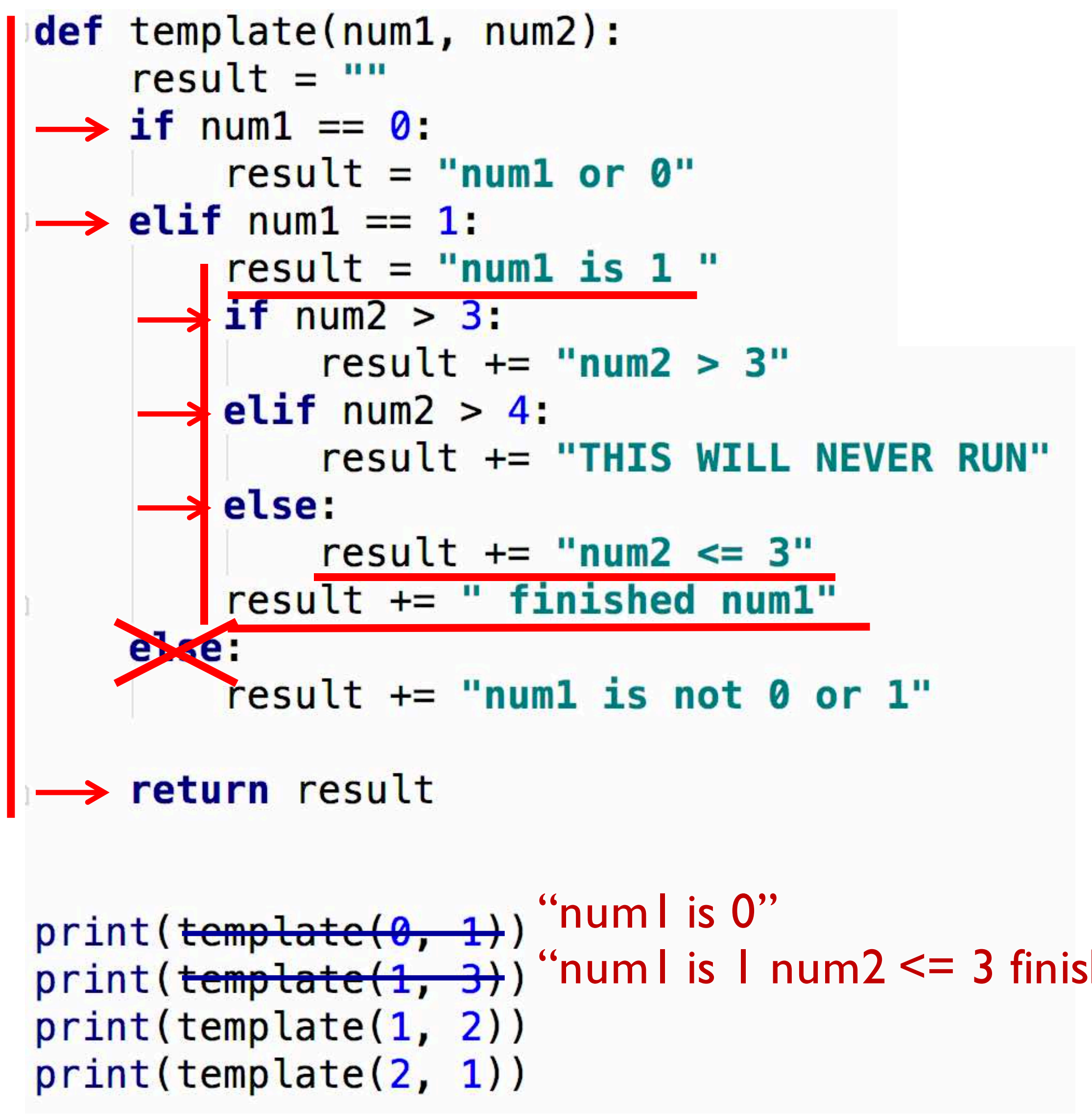
# Indentation Matters



```
def template(num1, num2):
    result = ""
    if num1 == 0:
        result = "num1 or 0"
    ~~elif~~ num1 == 1:
        result = "num1 is 1 "
        if num2 > 3:
            result += "num2 > 3"
        elif num2 > 4:
            result += "THIS WILL NEVER RUN"
        else:
            result += "num2 <= 3"
        result += " finished num1"
    ~~else~~:
        result += "num1 is not 0 or 1"

    return result


print(~~template(0, 1)~~) "num1 is 0"
print(template(1, 3))
print(template(1, 2))
print(template(2, 1))
```

This is another type of "nesting", and is usually referred to as "nested if-else statements"

# Indentation Matters

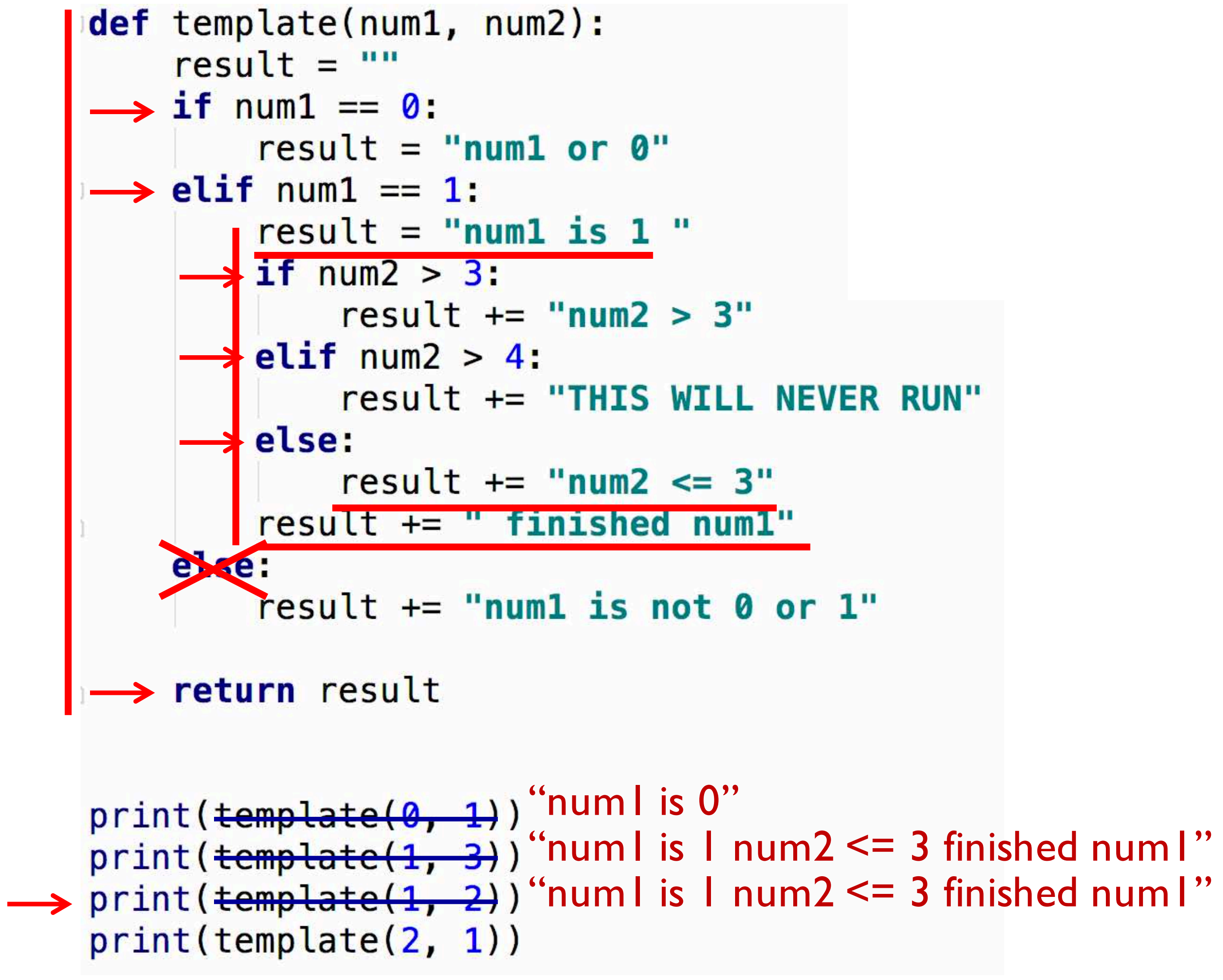


```
def template(num1, num2):
    result = ""
    if num1 == 0:
        result = "num1 or 0"
    elif num1 == 1:
        result = "num1 is 1 "
        if num2 > 3:
            result += "num2 > 3"
        elif num2 > 4:
            result += "THIS WILL NEVER RUN"
        else:
            result += "num2 <= 3"
        result += " finished num1"
    ~~else~~:
        result += "num1 is not 0 or 1"

    return result
```

print(~~template(0, 1)~~) “num1 is 0”
print(~~template(1, 3)~~) “num1 is 1 num2 <= 3 finished num1”

```
print(template(1, 2))
print(template(2, 1))
```

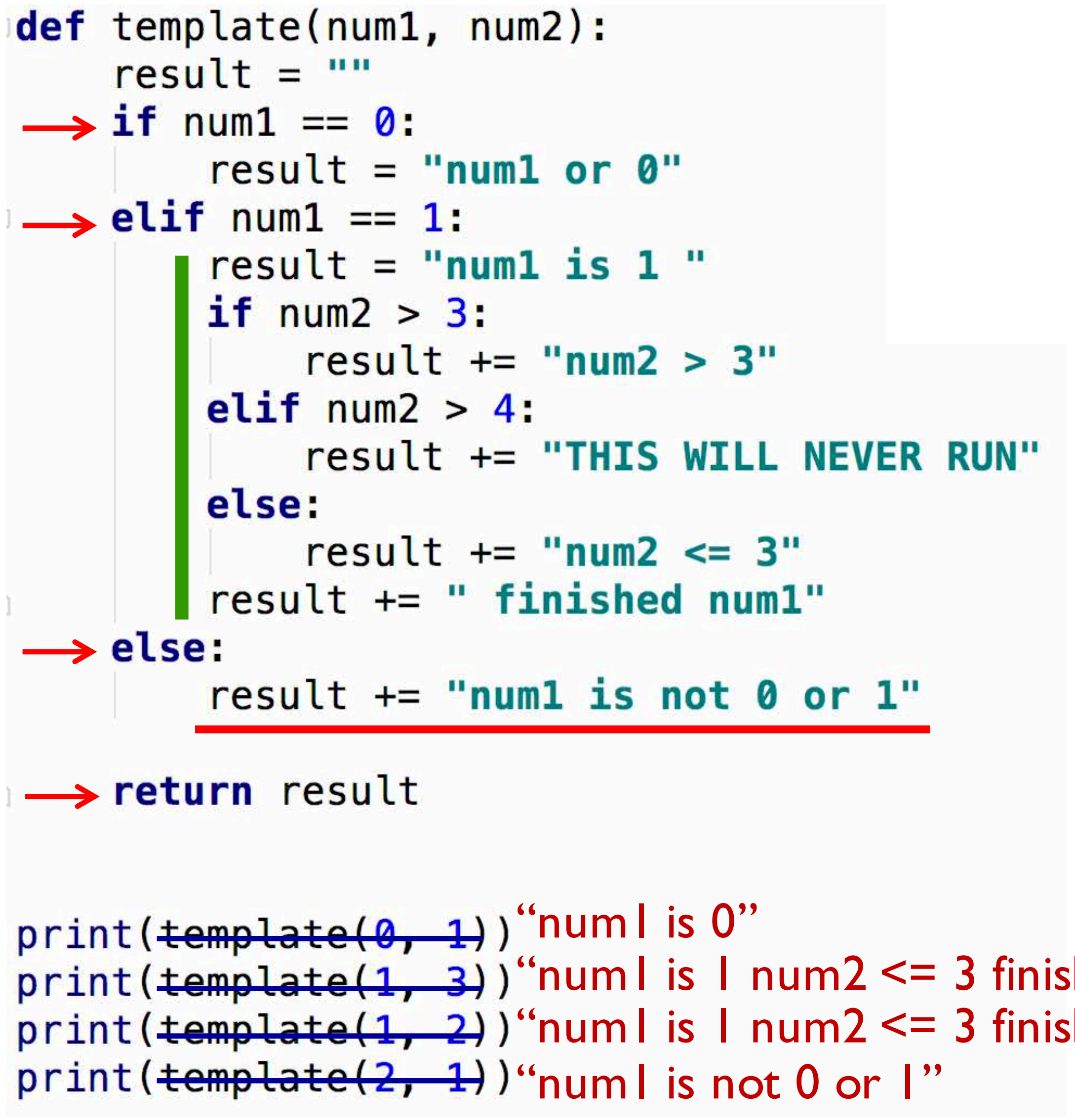
# Indentation Matters



```
def template(num1, num2):
    result = ""
    if num1 == 0:
        result = "num1 or 0"
    elif num1 == 1:
        result = "num1 is 1 "
        if num2 > 3:
            result += "num2 > 3"
        elif num2 > 4:
            result += "THIS WILL NEVER RUN"
        else:
            result += "num2 <= 3"
        result += " finished num1"
    else:
        result += "num1 is not 0 or 1"

    return result


print(template(0, 1))
print(template(1, 3))
print(template(1, 2))
print(template(2, 1))
```

“num1 is 0”
“num1 is 1 num2 <= 3 finished num1”
“num1 is 1 num2 <= 3 finished num1”

# Indentation Matters

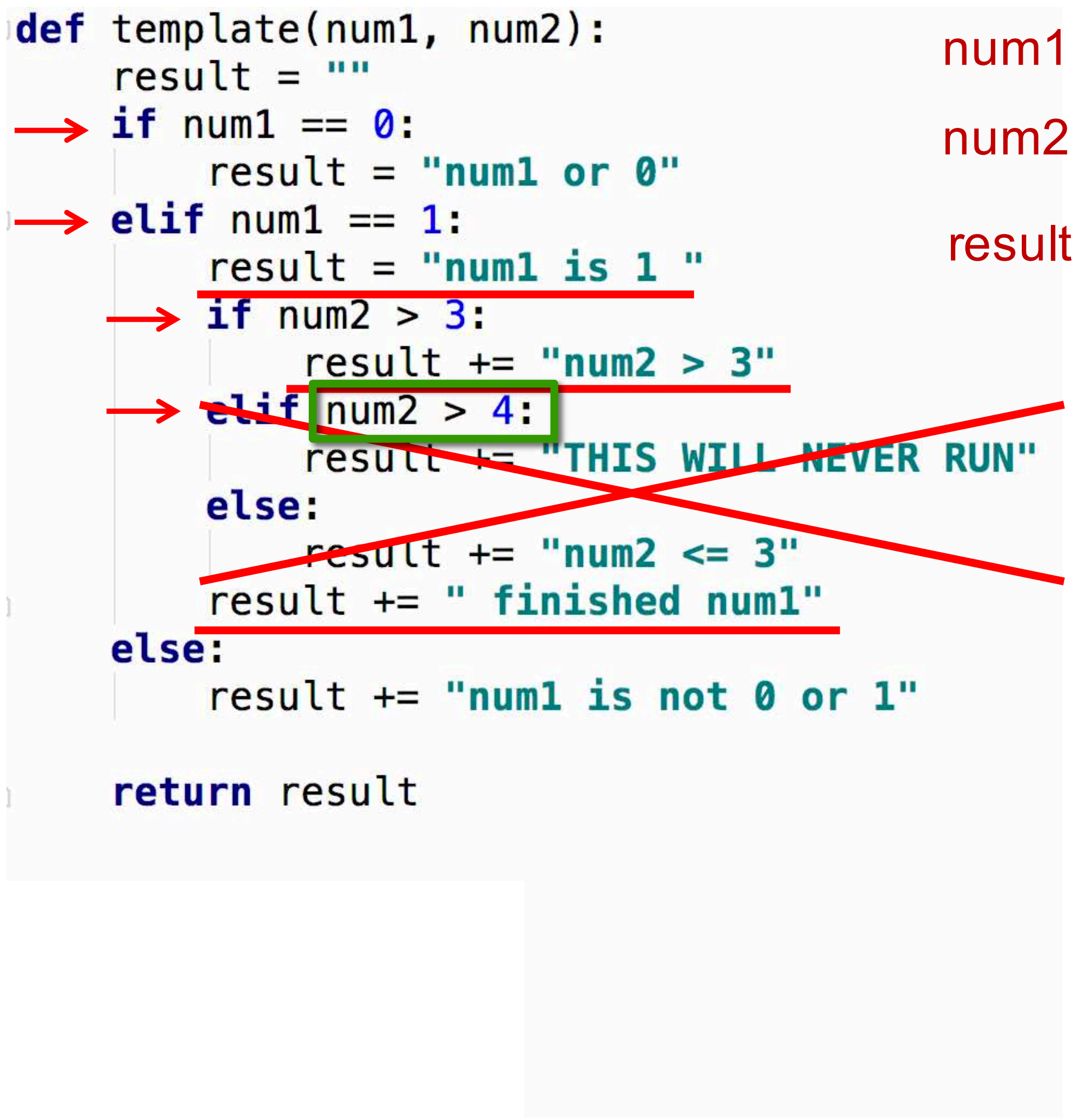


```
def template(num1, num2):
    result = ""
    if num1 == 0:
        result = "num1 or 0"
    elif num1 == 1:
        result = "num1 is 1 "
        if num2 > 3:
            result += "num2 > 3"
        elif num2 > 4:
            result += "THIS WILL NEVER RUN"
        else:
            result += "num2 <= 3"
        result += " finished num1"
    else:
        result += "num1 is not 0 or 1"

    return result


print(~~template(0, 1)~~)"num1 is 0"
print(~~template(1, 3)~~)"num1 is 1 num2 <= 3 finished num1"
print(~~template(1, 2)~~)"num1 is 1 num2 <= 3 finished num1"
print(~~template(2, 1)~~)"num1 is not 0 or 1"
```

Indentation groups things

if, elif, and else are mutually exclusive

# Unreachable statements



```
def template(num1, num2):
    result = ""
    if num1 == 0:
        result = "num1 or 0"
    elif num1 == 1:
        result = "num1 is 1 "
        if num2 > 3:
            result += "num2 > 3"
        elif num2 > 4:
            result += "THIS WILL NEVER RUN"
        else:
            result += "num2 <= 3"
        result += " finished num1"
    else:
        result += "num1 is not 0 or 1"

    return result
```

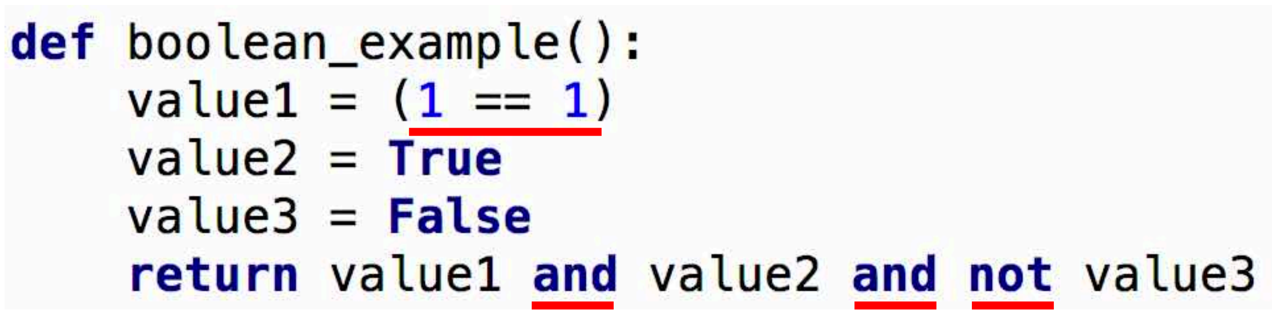
| | |
|---|---|
| num1 | 1 |
| num2 | 5 |
| result | "num1 is 1 num2 > 3 finished num1" |

print(template(1, 5))

# Programming TRAP

- Assignment statement x = "CS1111" x $\boxed{\text{CS1111}}$
- Boolean expression x $\underline{\ ==\ }$ "CS1111"

# Boolean Types



```python
def boolean_example():
    value1 = (1 == 1)
    value2 = True
    value3 = False
    return value1 and value2 and not value3
```

- `True` and `False` are both keywords and types in Python
  - Capitalization !!
- `not` is a keyword that negates a Boolean value
- The code above returns `True`

# Boolean Values and Calculations

| x | y | x and y |
|---|---|---|
| False | False | False |
| False | True | False |
| True | False | False |
| **True** | **True** | **True** |

| x | y | x or y |
|---|---|---|
| **False** | **False** | **False** |
| False | True | True |
| True | False | True |
| True | True | True |

- A boolean value must evaluate to true or false
- Two boolean values can be compared with **and** or **or**
- Use parentheses if you want to combine **and** or **or** to disambiguate; e.g., `(x and y) or z` or `x and (y or z)`

`x=F, y=T, z=T`

$\underbrace{\underbrace{(x \text{ and } y)}_{F} \text{ or } z}_{T}$ $\qquad$ $\underbrace{x \text{ and } \underbrace{(y \text{ or } z)}_{T}}_{F}$

- You can use any logical operators: **and** or **or** or **not**