Revision questions!

https://pollev.com/yuanzhizhuo862

Please copy and paste this URL to attend the Quiz!

MBSI Python Coding Workshop #3

Boolean Logic, Conditionals and Loops - Nicholas Huang and Ian Zhuo

Week 2 Recap

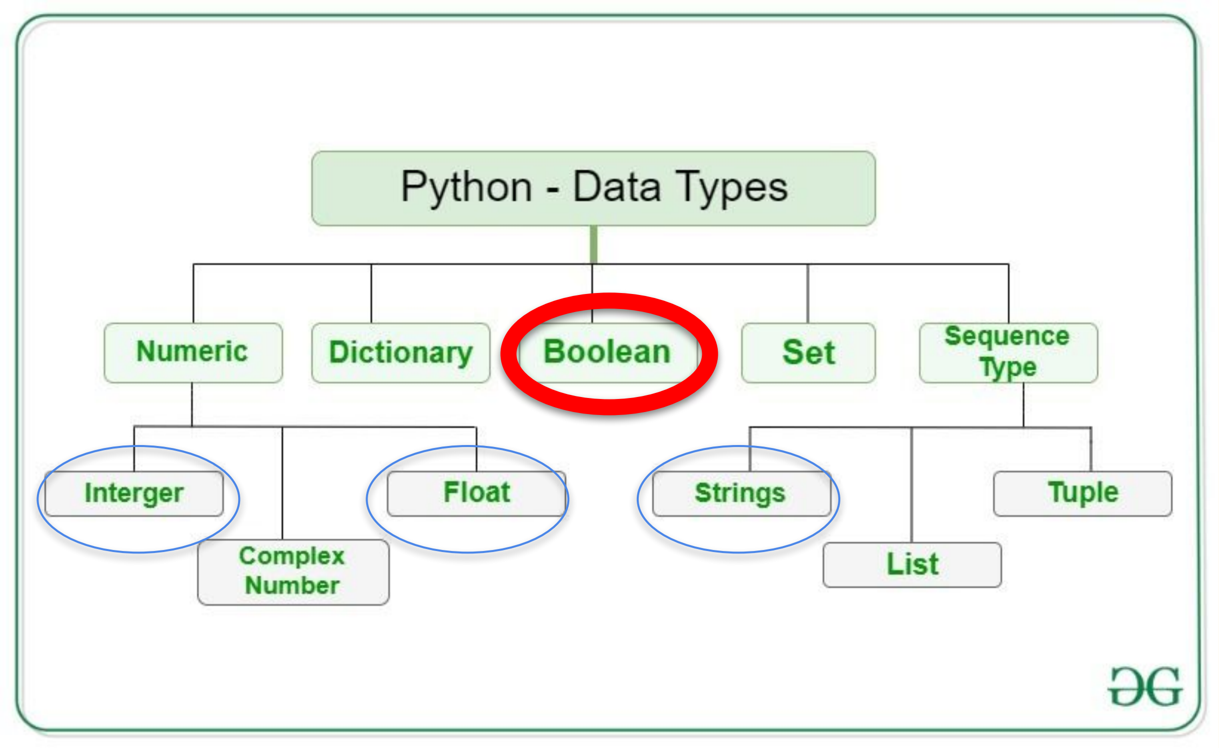
Week 2 Concepts	**Definition**	**Example**
Integers	Integers are whole numbers	5 is an integer
Floats	Floats are decimal numbers	5.0 is a float
Data type conversions	Converting between one data type and another	`x = str(5)` converts the integer 5 to the string “5”
Mathematical operators	`+, -, *, /, //, %, **`	`print(4 * 6)` outputs `24`
Updating variables	Overwriting a variable with another value	`x = 4` can be overwritten by `x = 8`
Incrementation	Overwriting a variable with reference to itself	`x += 5` is the same as `x = x + 5`

New Concepts

1. Boolean data type
    - Boolean expressions and comparison operators
    - Boolean expressions and Boolean logic
2. Pseudocode and flowcharts
3. Conditional statements
4. Part of Loops----While Loops

Booleans

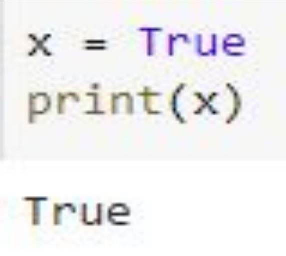
What we’ll cover today



Boolean Data Type

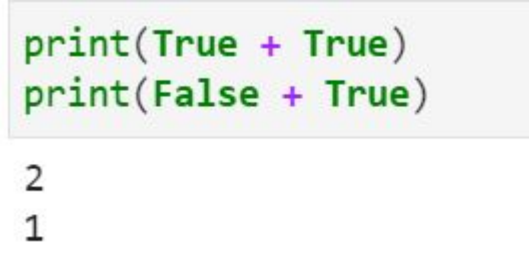
- A *Boolean value* can either be **`True`** or **`False`**
- Convert other data types into a Boolean with the **`bool()`** function
- Can be assigned to variables but it’s not possible to assign a value to Boolean.

George Boole (1815-1864)



```
x = True
print(x)
```

```
True
```



```
print(True + True)
print(False + True)
```

```
2
1
```

Why are booleans useful?

- Booleans represent the *truth* of a *statement* or *expression*
- Is this statement true?

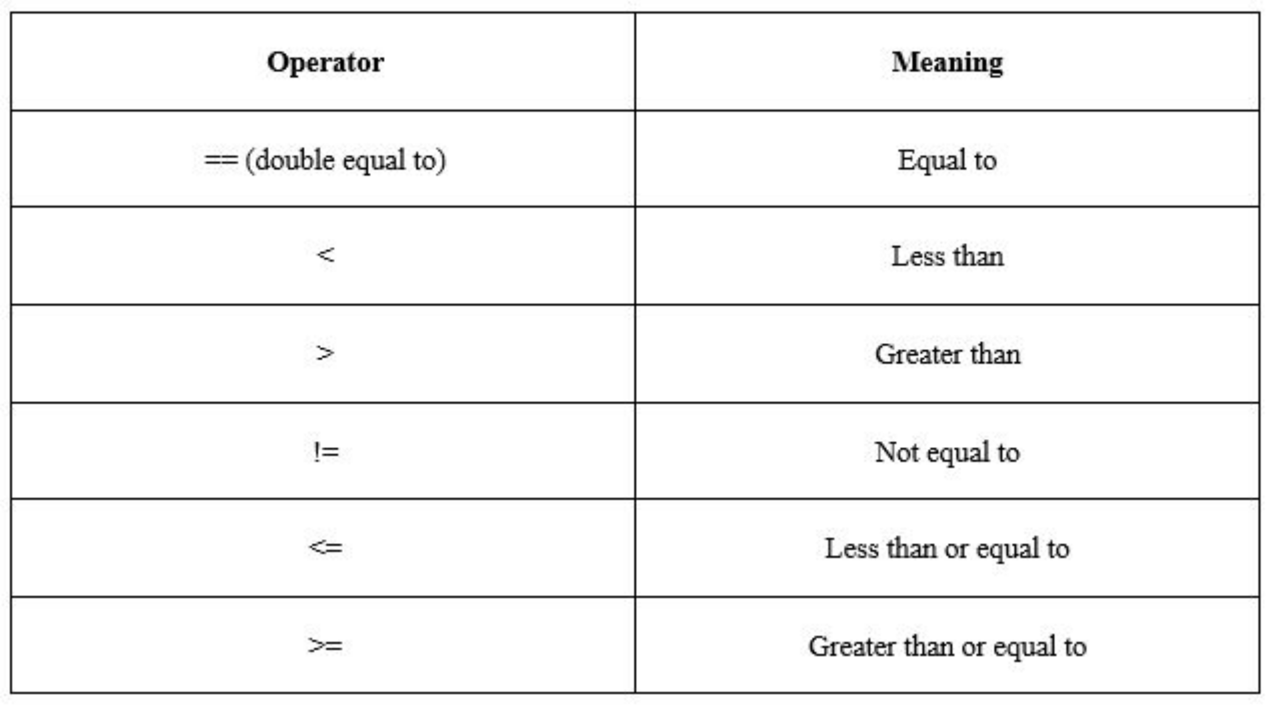
  2 < 5

- What about this statement?

  15 > 3 * 5

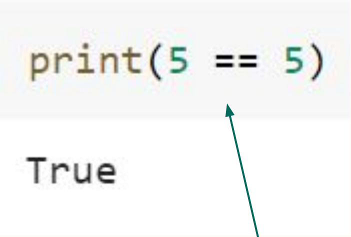
Comparison Operators

With *comparison operators*, you can write a boolean expression that compares the values of two objects and prompts the computer to compute whether it is `True` or `False`.



Operator	Meaning
== (double equal to)	Equal to
<	Less than
>	Greater than
!=	Not equal to
<=	Less than or equal to
>=	Greater than or equal to

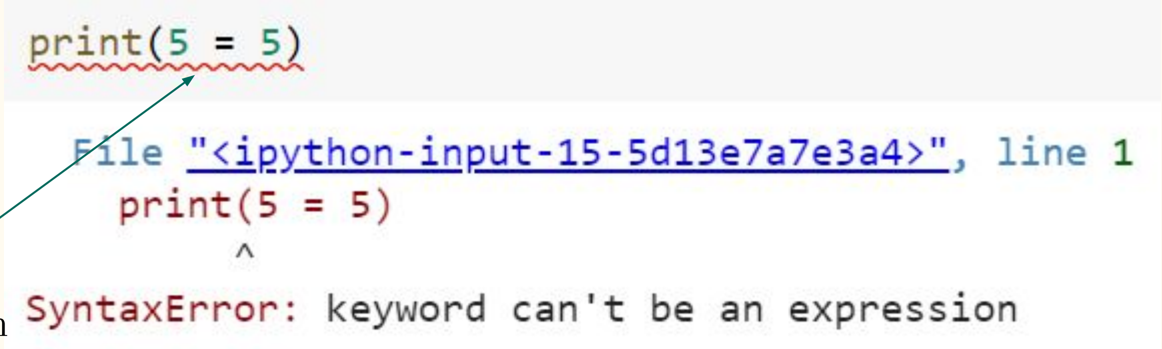
Comparison Operators Examples



```
print(5 == 5)
```

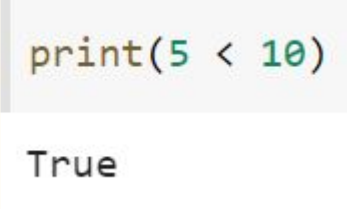
```
True
```

Remember the difference between '=' and '=='!



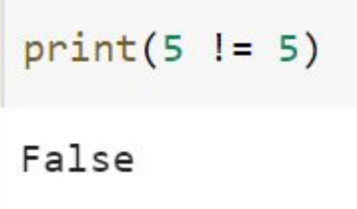
```
print(5 = 5)
```

```
  File "<ipython-input-15-5d13e7a7e3a4>", line 1
    print(5 = 5)
         ^
SyntaxError: keyword can't be an expression
```



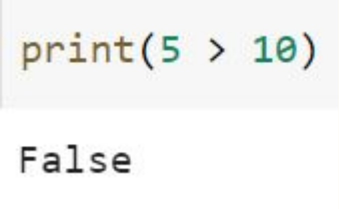
```
print(5 < 10)
```

```
True
```



```
print(5 != 5)
```

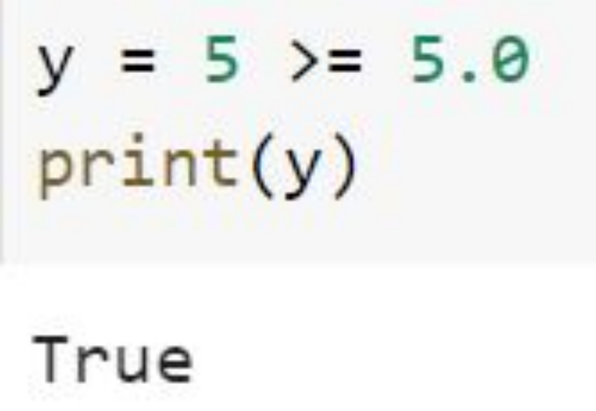
```
False
```



```
print(5 > 10)
```

```
False
```

Assigning Boolean expressions to variables



```
y = 5 >= 5.0
print(y)
```

```
True
```

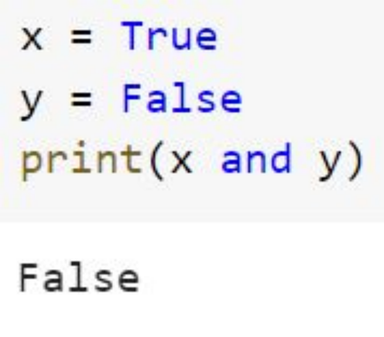
Logical Operators

- A type of Boolean expression that deals with boolean values
- 3 *logical operators*:
    - `and`
        - `True` if both booleans are `True`
        - `False` otherwise
    - `or`
        - `True` if at least one boolean is `True`
        - `False` otherwise
    - `not`
        - `True` if boolean is `False`
        - `False` if boolean is `True`

AND returns True only if both inputs are True.

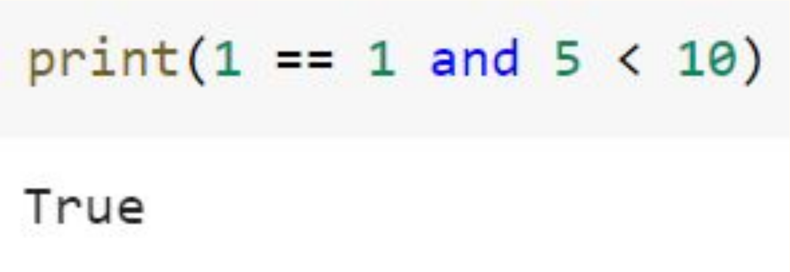
Truth table:

and	True	False
True	True	False
False	False	False



```
x = True
y = False
print(x and y)
```

```
False
```



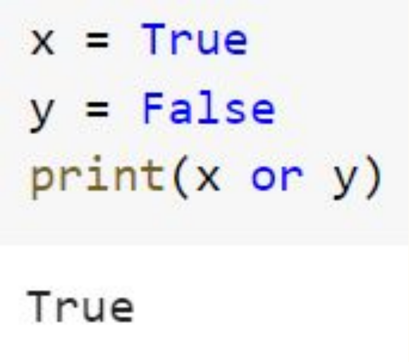
```
print(1 == 1 and 5 < 10)
```

```
True
```

OR returns True if at least one input is True.

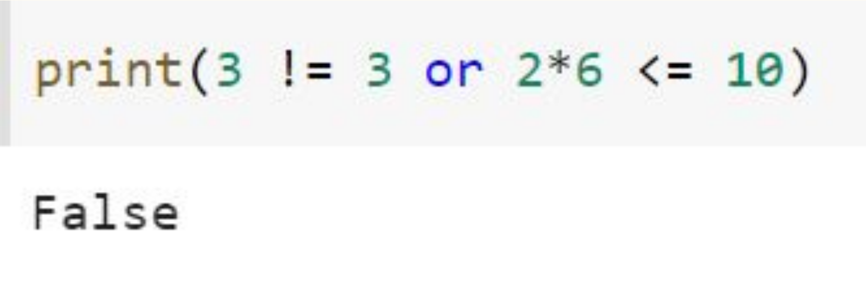
Truth table:

or	True	False
True	True	True
False	True	False



```
x = True
y = False
print(x or y)
```

```
True
```



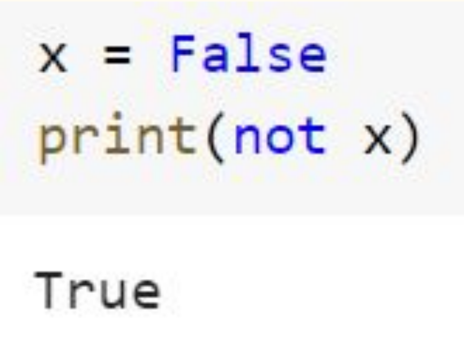
```
print(3 != 3 or 2*6 <= 10)
```

```
False
```

NOT returns the negation of the input.

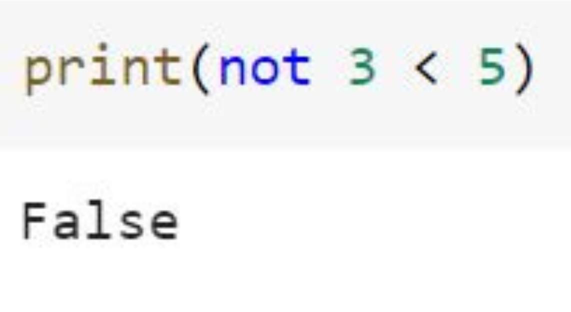
Truth table:

x	not x
True	False
False	True



```
x = False
print(not x)
```

```
True
```



```
print(not 3 < 5)
```

```
False
```

Principles of Algorithms

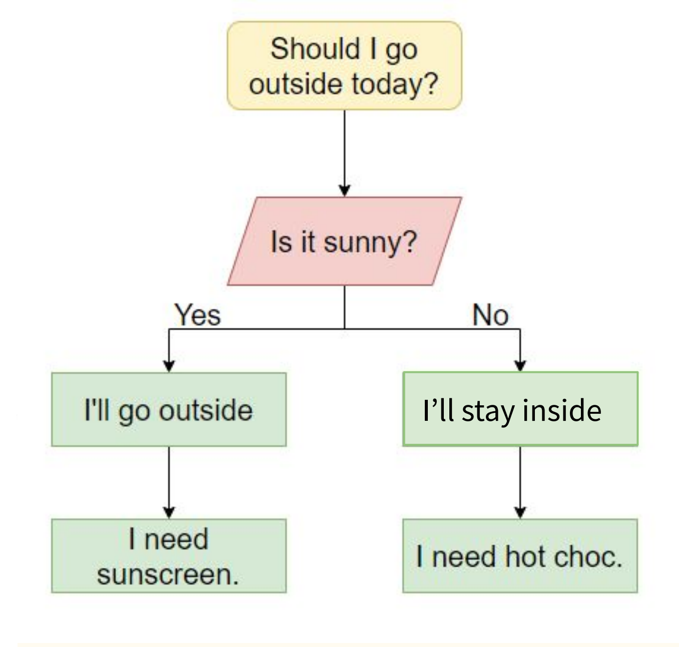
Should I go outside today?

- How do I systematically make a decision?

A flowchart?

Written down explicitly?

```
# A program to decide if I should go out today.

Is it sunny at the moment?
If yes:
     I should go outside.
     I should also put on some sunscreen.
If no:
     I shouldn’t go outside.
     I’ll make a nice cup of hot choc.
```



Should I go outside today?

Is it sunny?

Yes

No

I'll go outside

I’ll stay inside

I need sunscreen.

I need hot choc.

Principles of Algorithms

Coding: Expectation

- SYNTAX
- CONFUSION

Coding: Reality

- SYNTAX
- PROBLEM SOLVING
- CONFUSION

It hurt itself in its confusion!

Pseudocode

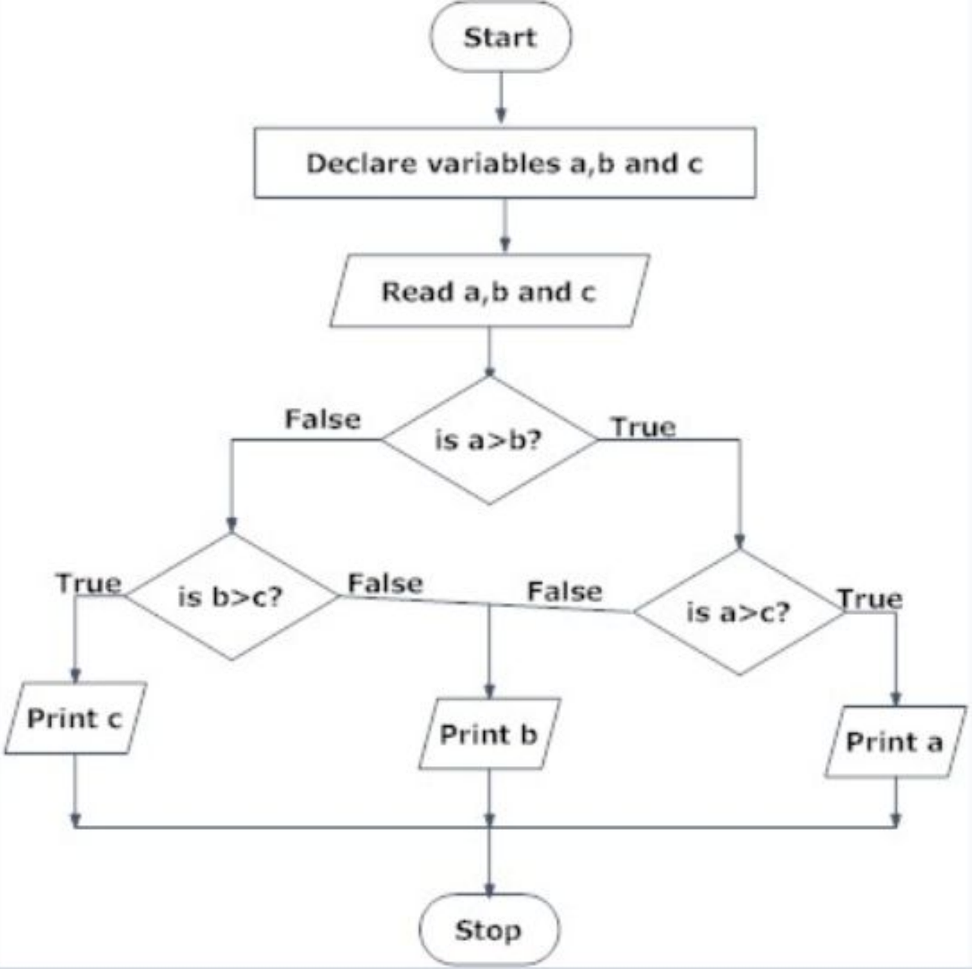
- Plain language description of what your code will do
- Doesn’t use syntax specific to any programming language but can be structured like actual code
- Not executable
- Just let you know what you will get from your plan

```
# A program to decide if I should go out today.

Is it sunny at the moment? If yes:
     I should go outside.
     I should also put on some sunscreen.
If no:
     I shouldn’t go outside.
     I’ll make a nice cup of hot choc.
```

Flowchart

This *flowchart* describes a program that:

- Takes in 3 numbers stored in variables `a`, `b` and `c`
- Compares their values
- Outputs the largest number

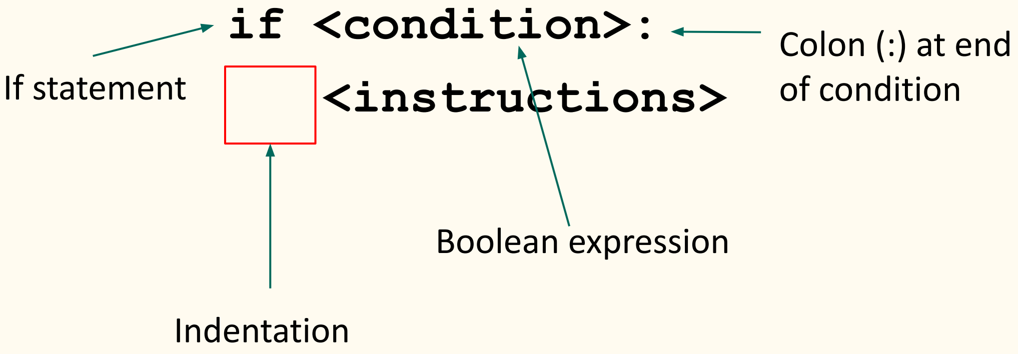


Flowchart to find the largest among three numbers.

Conditional statements

Conditional Statements (`if`)

- *Conditional statements* are used for decision-making within a program depending on whether some *condition* is met
- The most basic conditional is an `if` statement:



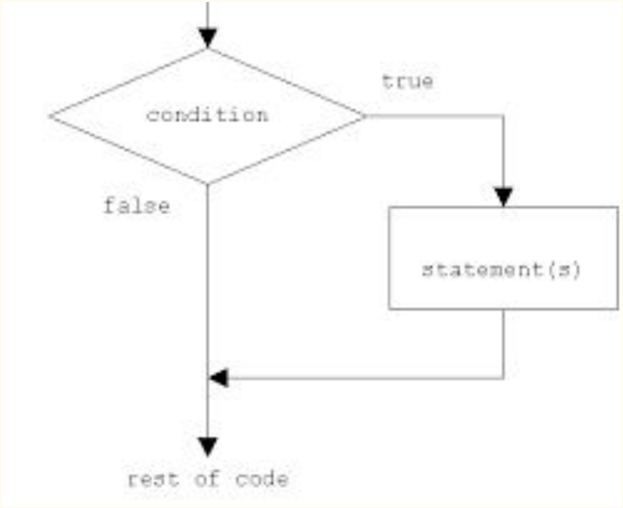
```
if <condition>:
    <instructions>
```

If statement

Colon (:) at end of condition

Boolean expression

Indentation

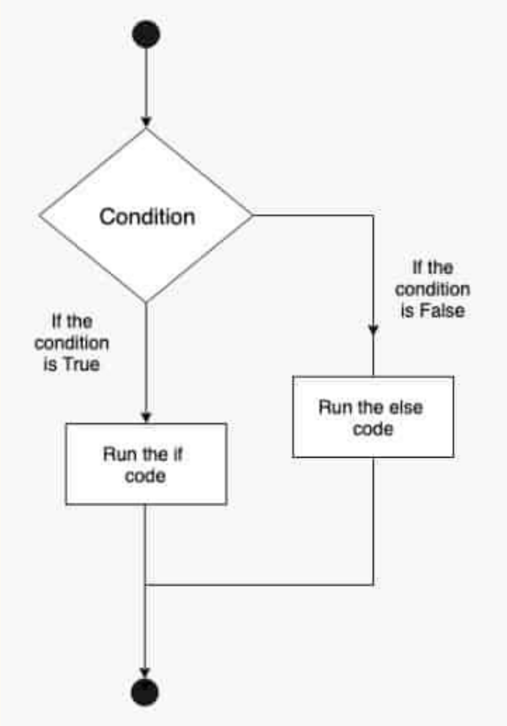


Conditional Statements (`else`)

- If we want to specify what to do when the condition is `false` we can add an `else` statement:

```
if <condition>:
   <instruction 1>
else:
   <instruction 2>
```

Else statement

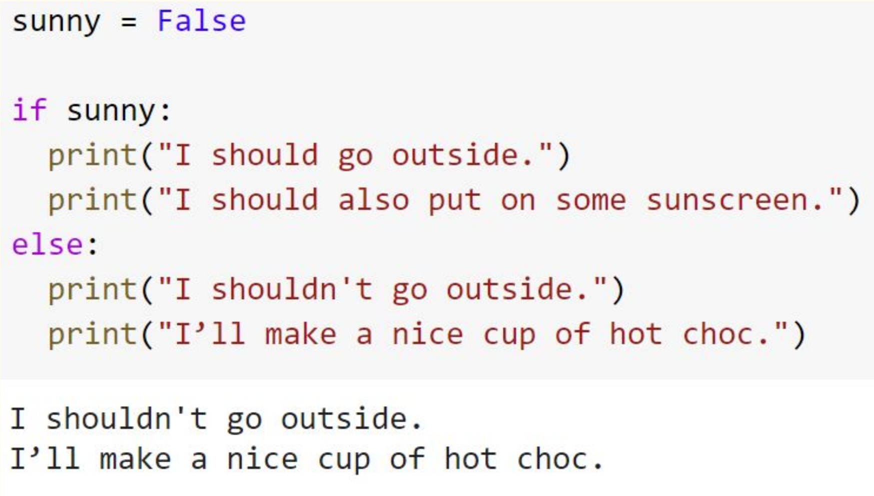


Conditional Statements Example

Pseudocode:

```
# A program to decide if I should go out today.

Is it sunny at the moment? If yes:
     I should go outside.
     I should also put on some sunscreen.
If no:
     I shouldn’t go outside.
     I’ll make a nice cup of hot choc.
```

Code:



```python
sunny = False

if sunny:
  print("I should go outside.")
  print("I should also put on some sunscreen.")
else:
  print("I shouldn't go outside.")
  print("I’ll make a nice cup of hot choc.")
```

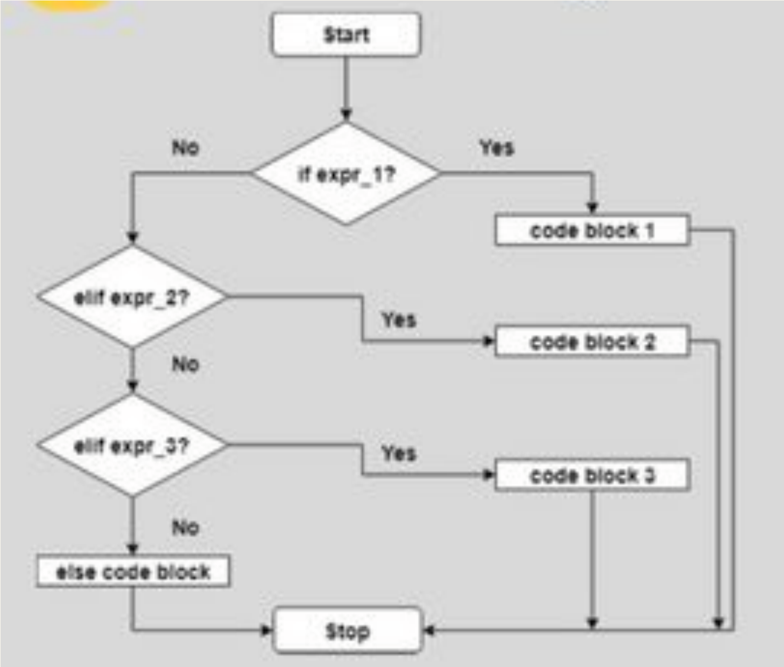
```
I shouldn't go outside.
I’ll make a nice cup of hot choc.
```

Conditional Statements (`elif`)

- We can use `elif` statements between `if` and `else` statements if we have more than one condition:

```
if <condition 1>:
    <instruction 1>
elif <condition 2>:
    <instruction 2>
elif <condition 3>:
    <instruction 3>
else:
    <instruction 4>
```

Elif statements



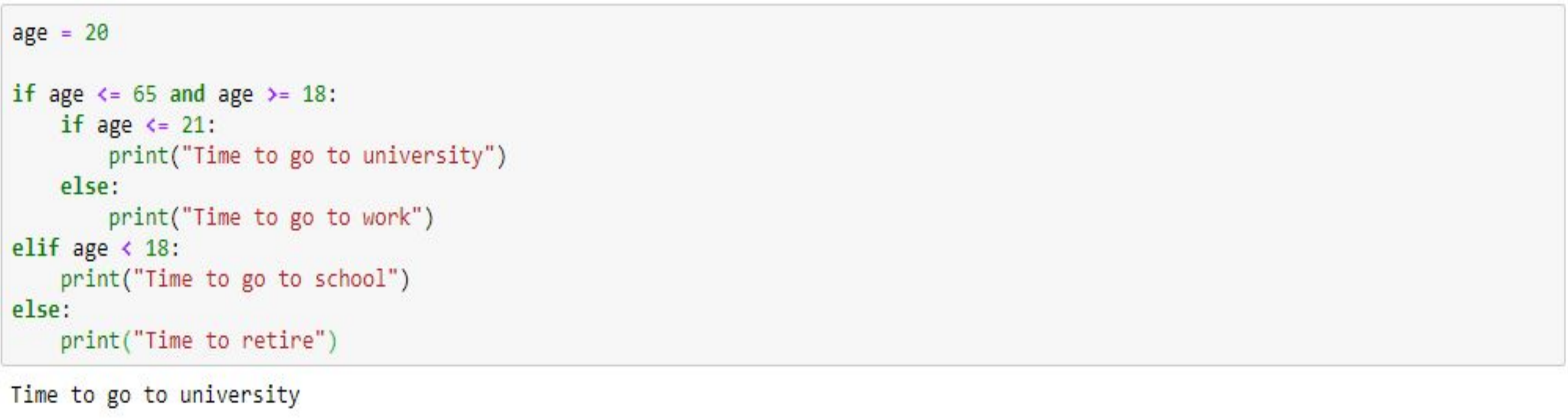
Nested Conditional Statements

- We can also put conditional statements inside conditional statements if there are multiple decision paths. This is called *nesting*.

```
if <condition 1>:
    if <condition 2>:
        <instruction 1>
    else:
        <instruction 2>
else:
    <instruction 3>
```

Nested conditional statements

Nested Conditional example



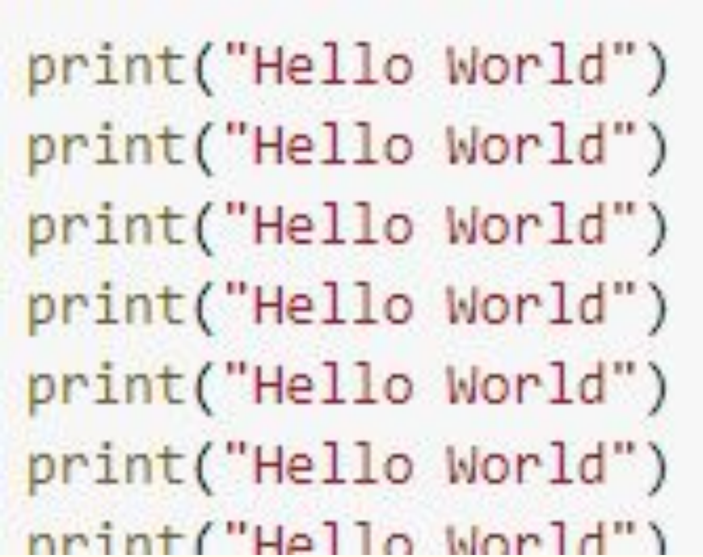
```
age = 20

if age <= 65 and age >= 18:
    if age <= 21:
        print("Time to go to university")
    else:
        print("Time to go to work")
elif age < 18:
    print("Time to go to school")
else:
    print("Time to retire")
```

```
Time to go to university
```

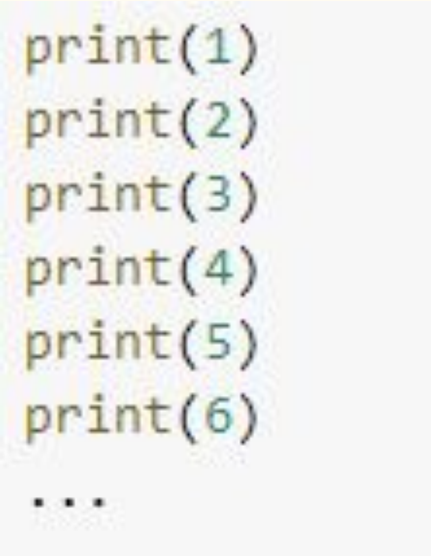
Loops

Say you want to do the following...

- Print “`Hello World`” 20 times
- Print the numbers from 1 to 100



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



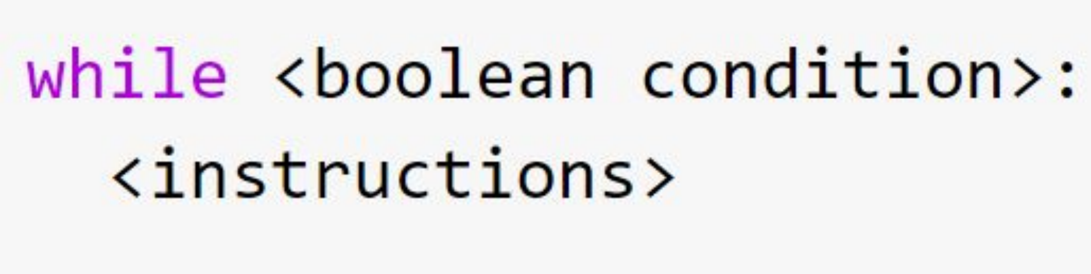
```
print(1)
print(2)
print(3)
print(4)
print(5)
print(6)
...
```

Loops overview

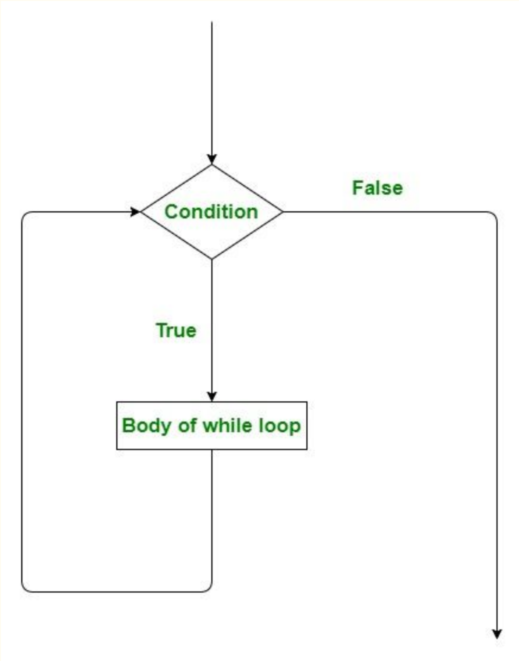
- An efficient way to **repeat** code (that doesn’t need copy-paste)
- Two types of loops:
  - **While loops**
  - **For loops**

while loop

- Repeats same chunk of code **while** some condition is satisfied
  - Every time the loop runs, we call it an “**iteration**”

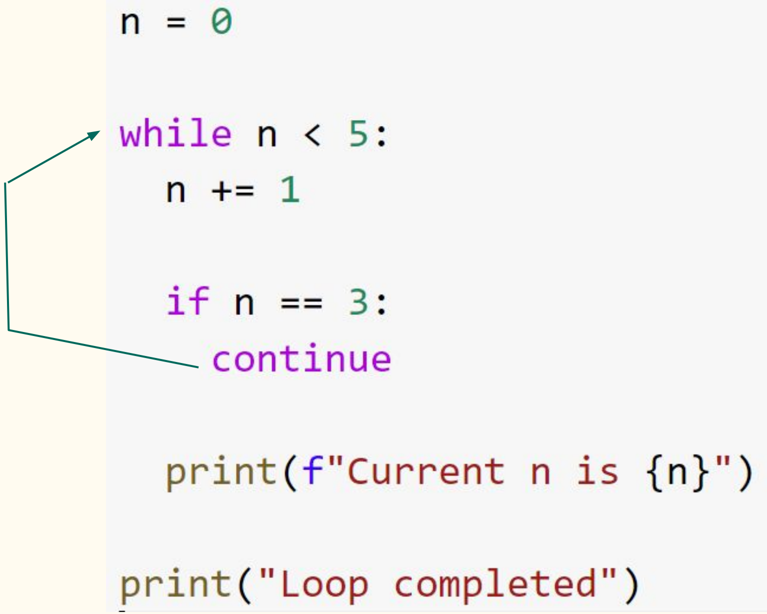


```
while <boolean condition>:
  <instructions>
```

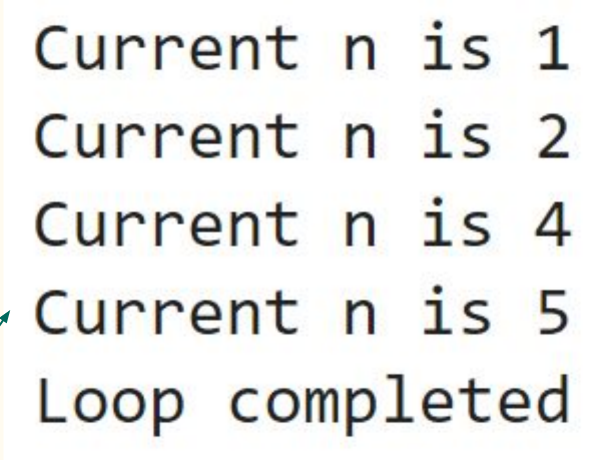


continue

- continue forces the loop to restart from the top
    - Useful if you want to skip something



```python
n = 0

while n < 5:
  n += 1

  if n == 3:
    continue

  print(f"Current n is {n}")

print("Loop completed")
```

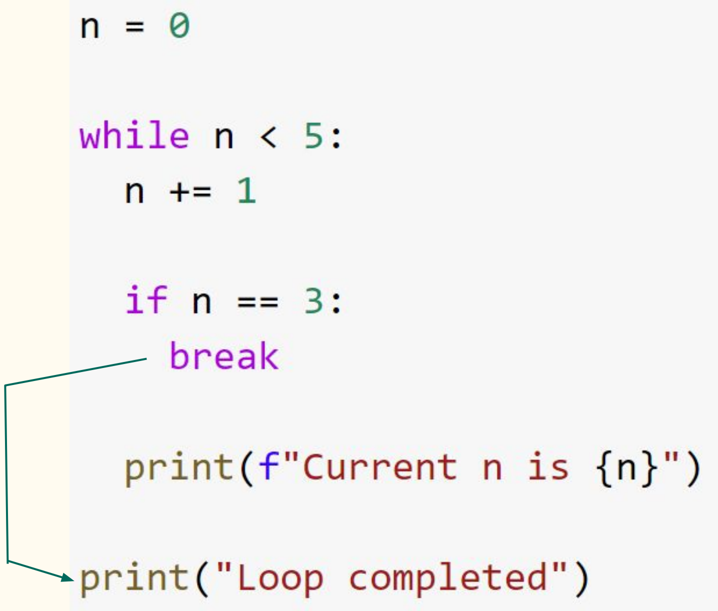


```
Current n is 1
Current n is 2
Current n is 4
Current n is 5
Loop completed
```

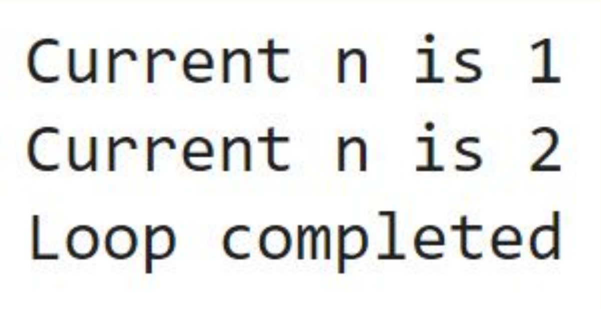
This is the output!

break

- `break` forces an exit from the loop
    - Useful if you want to terminate the loop early



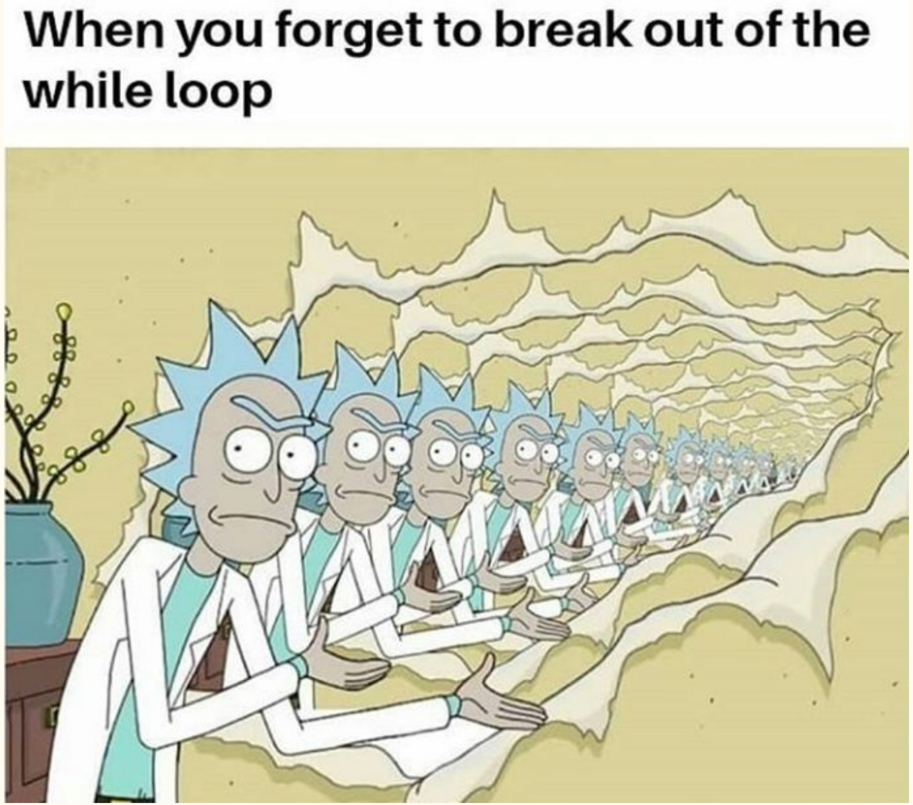
```python
n = 0

while n < 5:
  n += 1

  if n == 3:
    break

  print(f"Current n is {n}")

print("Loop completed")
```



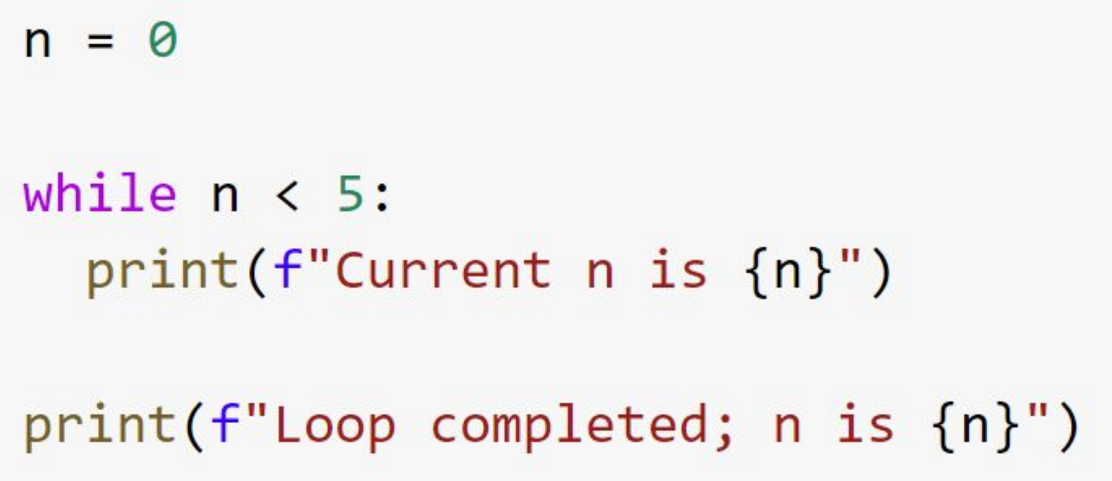
```
Current n is 1
Current n is 2
Loop completed
```

Be careful of infinite loops!!

An infinite loop can be dangerous if it never blocks or sleeps. This can take the CPU to near 100% utilization and prevent other programs from running very well.



Be careful of infinite loops!!

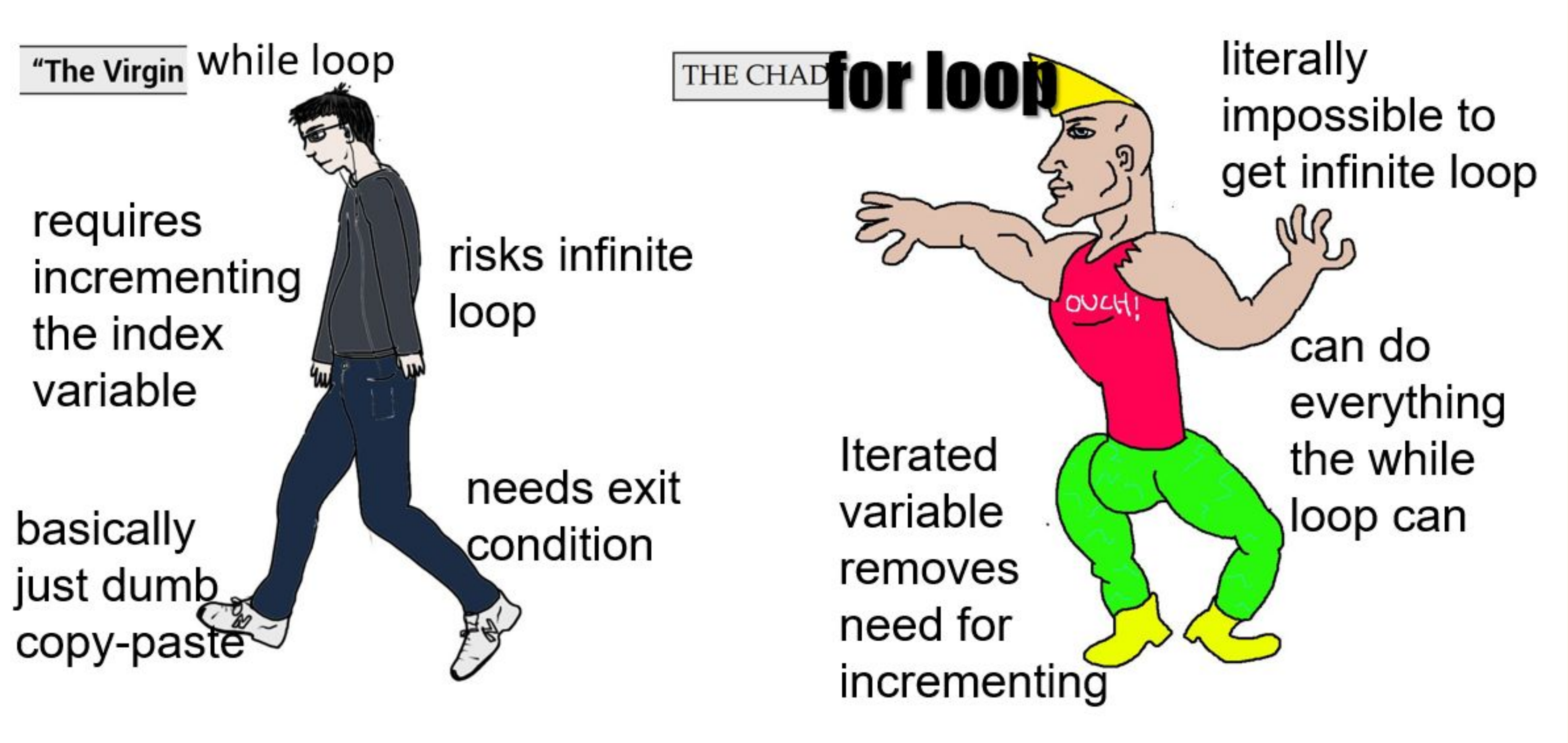


```
n = 0

while n < 5:
  print(f"Current n is {n}")

print(f"Loop completed; n is {n}")
```

Some strategies to avoid infinite loops:

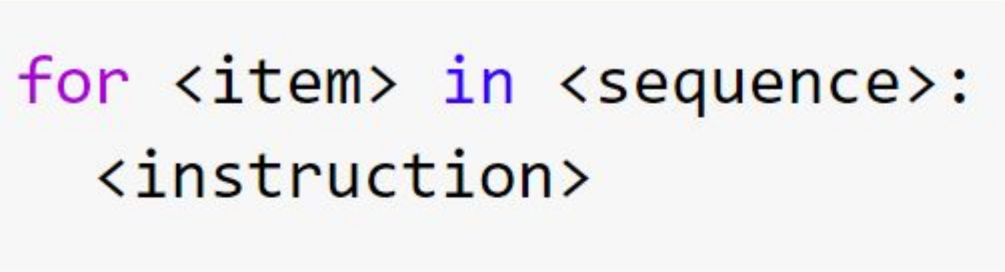
- Make your loop condition well-defined
- Ensure you do any appropriate incrementing
- Ensure you have a “`break`” condition somewhere

For Loop


"The Virgin while loop
requires incrementing the index variable
risks infinite loop
needs exit condition
basically just dumb copy-paste
THE CHAD for loop
literally impossible to get infinite loop
OUCH!
can do everything the while loop can
Iterated variable removes need for incrementing

for loop

- Repeats same chunk of code **for** each item in a sequence
    - A sequence, or an “iterable”, can be a list, tuple, string etc.
    - Sequence length specifies total number of iterations
    - The <item> becomes an “iterator” variable that can be used inside the loop



```
for <item> in <sequence>:
   <instruction>
```

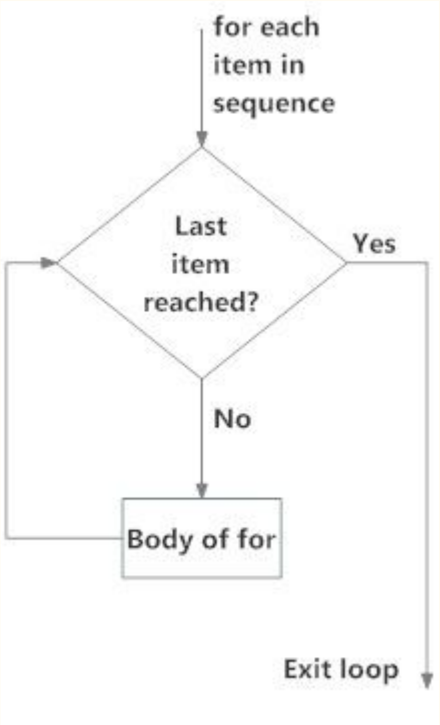
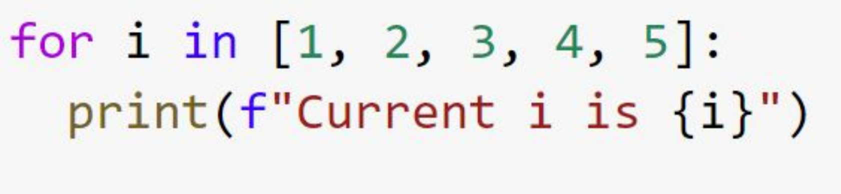
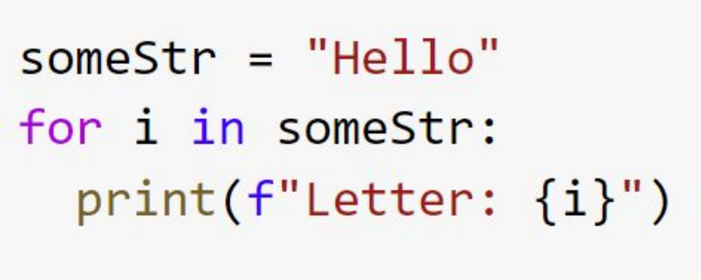


Fig: operation of for loop

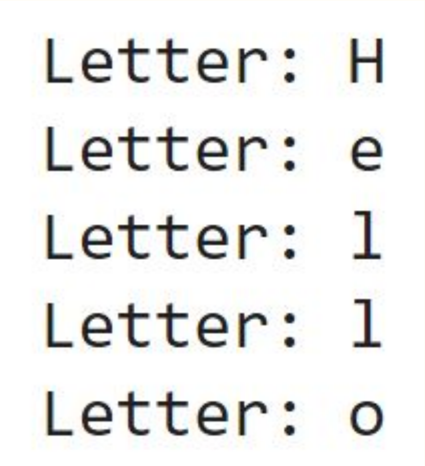
for loops using iterables



```
for i in [1, 2, 3, 4, 5]:
  print(f"Current i is {i}")
```



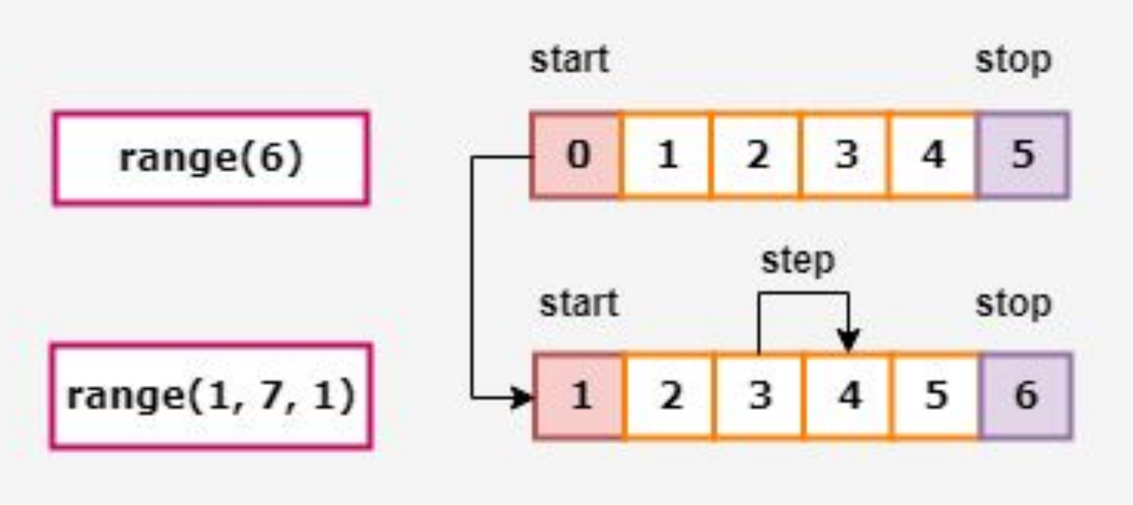
```
someStr = "Hello"
for i in someStr:
  print(f"Letter: {i}")
```



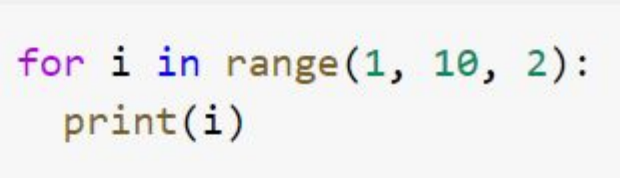
```
Letter: H
Letter: e
Letter: l
Letter: l
Letter: o
```

`range()`

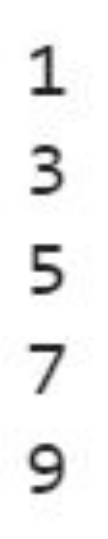
- Generates a sequence of numbers
- `range(n)` generates a sequence from 0 to (`n-1`) with a step-size of 1
- `range(a, n, s)` generates a sequence from starting value `a` to (`n-1`) with a step-size of `s`.



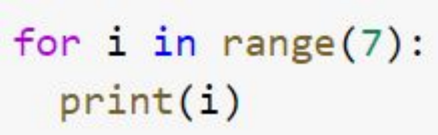
`range()` interactive examples



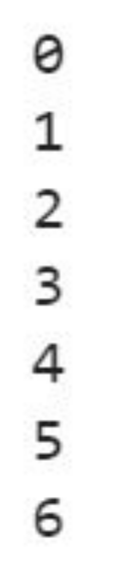
```
for i in range(1, 10, 2):
  print(i)
```



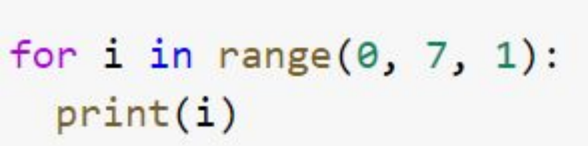
```
1
3
5
7
9
```



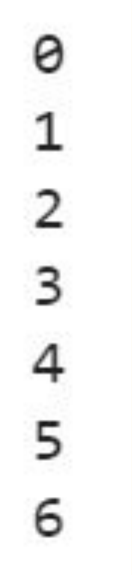
```
for i in range(7):
  print(i)
```



```
0
1
2
3
4
5
6
```

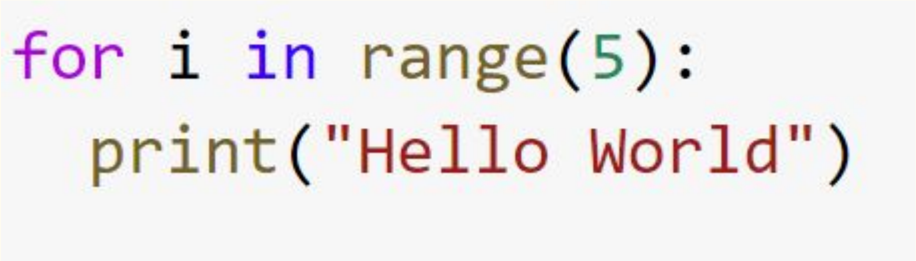


```
for i in range(0, 7, 1):
  print(i)
```

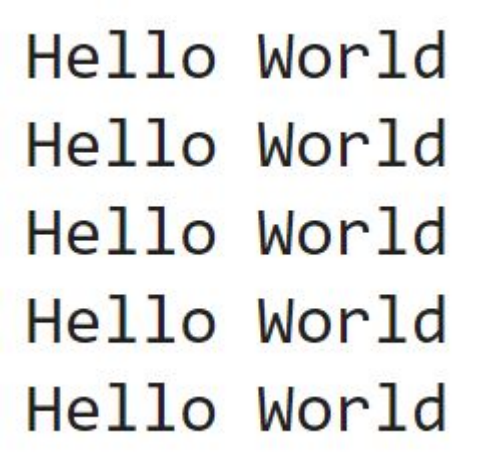


```
0
1
2
3
4
5
6
```

Using `range()` to count

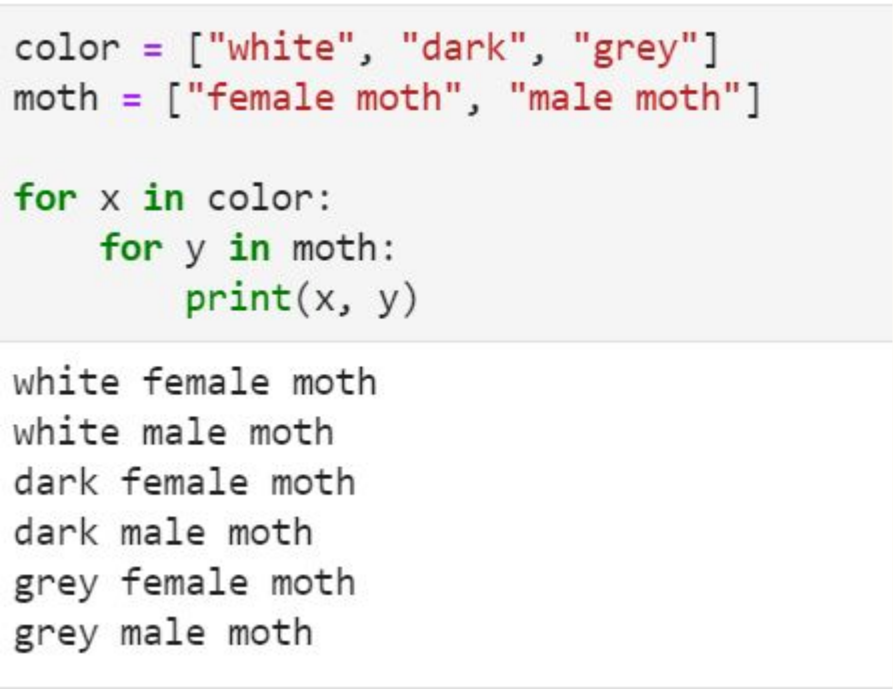


```
for i in range(5):
  print("Hello World")
```



```
Hello World
Hello World
Hello World
Hello World
Hello World
```

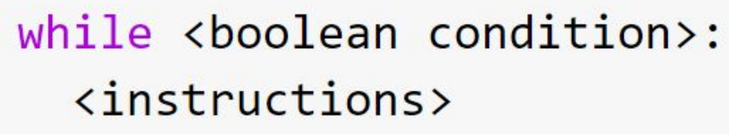
Nested loops example



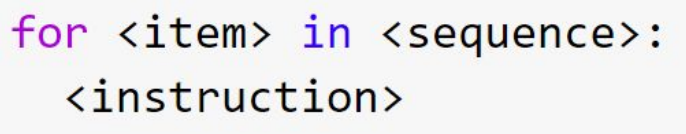
```python
color = ["white", "dark", "grey"]
moth = ["female moth", "male moth"]

for x in color:
    for y in moth:
        print(x, y)
```

```
white female moth
white male moth
dark female moth
dark male moth
grey female moth
grey male moth
```

Summary



```
while <boolean condition>:
  <instructions>
```



```
for <item> in <sequence>:
  <instruction>
```

	`while` loops	`for` loops
Loop is executed ...	… whenever a condition is satisfied	… over the items of a predetermined sequence or iterable (list/tuple/string etc.)
Use when ...	… **you don’t know how times to run the loop (ie. iterations)** but you do know when to stop the loop (ie. the condition)	… **you know exactly how many times to run the loop**

mini_Project: Assessing Lindsay Brown for Pulmonary Embolism

mini_Project: PE Diagnostic Assessment

mini_Project brief:

- Using information from RACGP guidelines, we will create a function that uses *Booleans* and *conditional statements* to:
  - Calculate and input the Wells score, PERC rule and D-dimer test results
  - Output a decision to exclude a pulmonary embolism or order imaging for definitive diagnosis

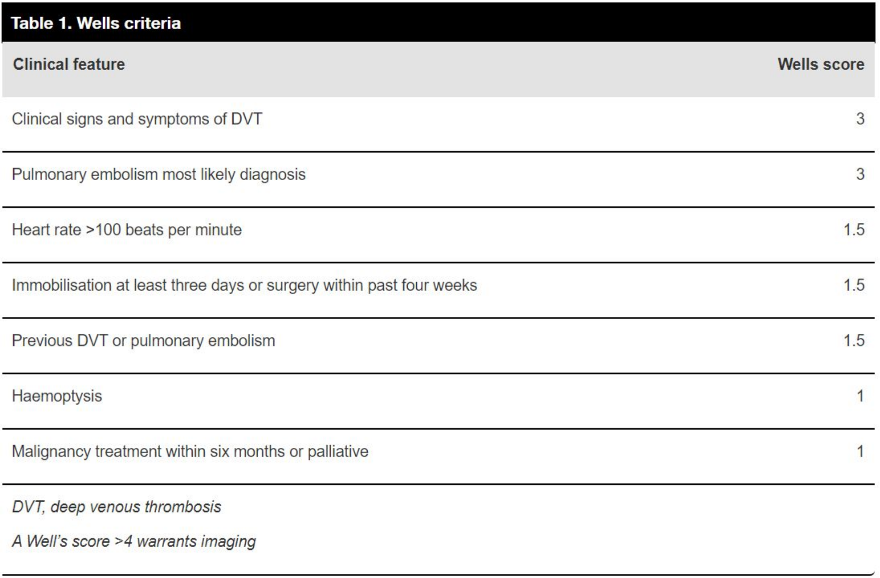
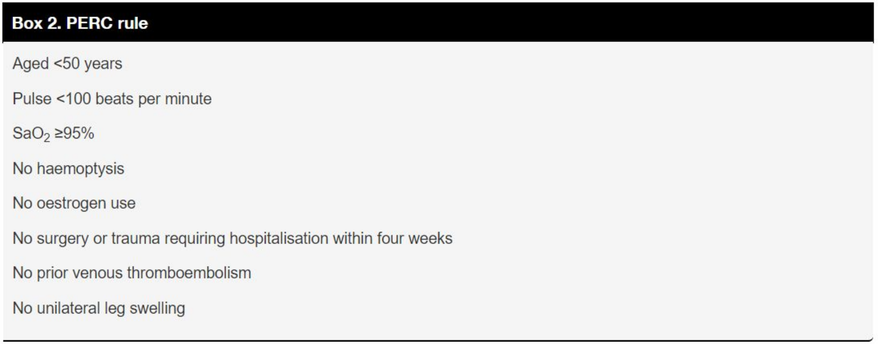


Table 1. Wells criteria

Clinical feature	Wells score
Clinical signs and symptoms of DVT	3
Pulmonary embolism most likely diagnosis	3
Heart rate >100 beats per minute	1.5
Immobilisation at least three days or surgery within past four weeks	1.5
Previous DVT or pulmonary embolism	1.5
Haemoptysis	1
Malignancy treatment within six months or palliative	1

DVT, deep venous thrombosis

A Well's score >4 warrants imaging



Box 2. PERC rule

Aged <50 years

Pulse <100 beats per minute

SaO_2 ≥95%

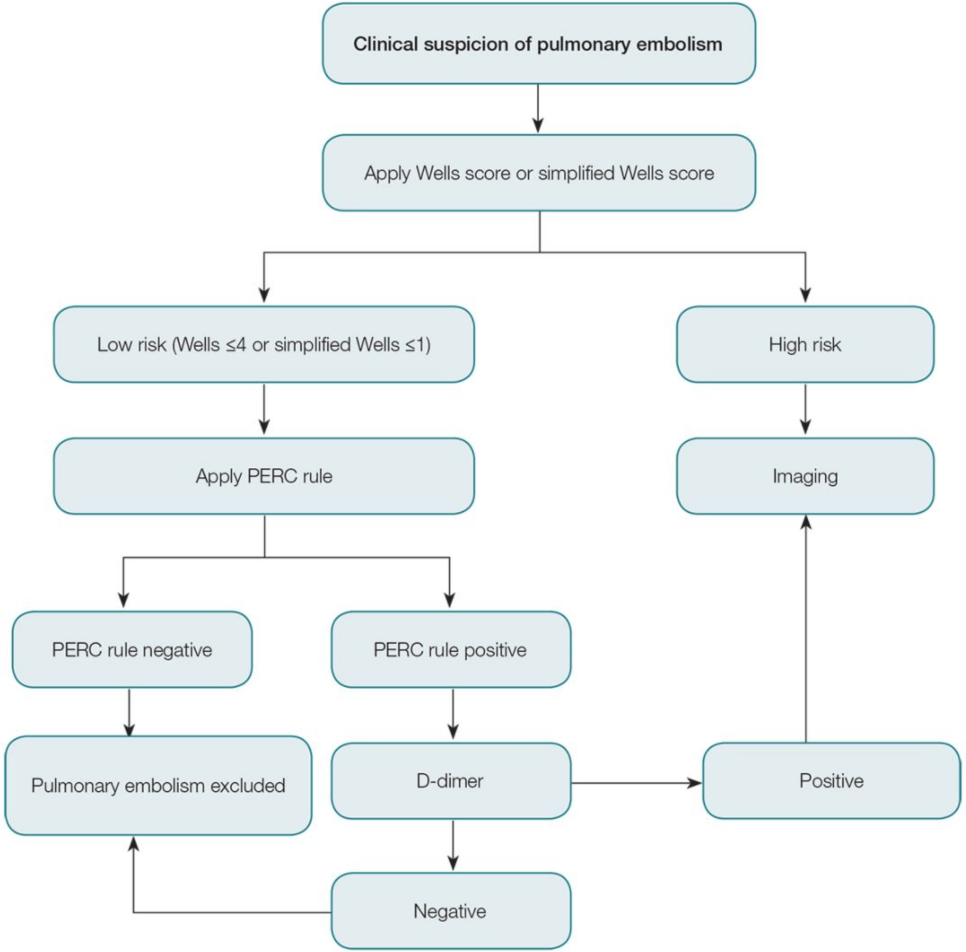
No haemoptysis

No oestrogen use

No surgery or trauma requiring hospitalisation within four weeks

No prior venous thromboembolism

No unilateral leg swelling



Clinical suspicion of pulmonary embolism

Apply Wells score or simplified Wells score

Low risk (Wells ≤4 or simplified Wells ≤1)

High risk

Apply PERC rule

Imaging

PERC rule negative

PERC rule positive

Pulmonary embolism excluded

D-dimer

Positive

Negative

Figure 1. Approach to investigation of pulmonary embolism

PERC, Pulmonary Embolism Rule-out Criteria

Breakout Time!

FEEDBACK FORM:

https://forms.gle/1GnsHhYUav7D281F8