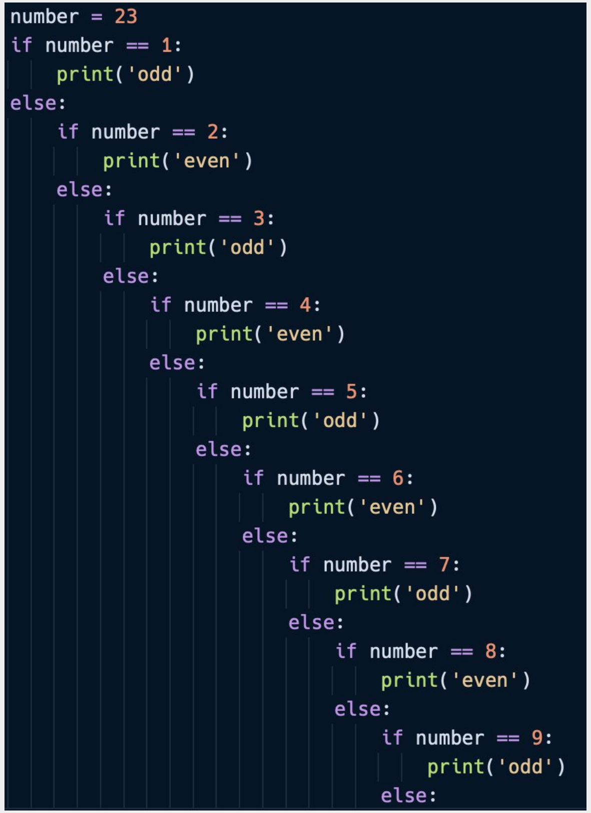
Day 3: For and While Loops, and Lists

July 12th 2023

bit.ly/3O7T5nL

Previous Day Clarifications

- getting unstuck: link on main page of website! some suggestions
- if/else/elif review: grade scale



```
number = 23
if number == 1:
    print('odd')
else:
    if number == 2:
        print('even')
    else:
        if number == 3:
            print('odd')
        else:
            if number == 4:
                print('even')
            else:
                if number == 5:
                    print('odd')
                else:
                    if number == 6:
                        print('even')
                    else:
                        if number == 7:
                            print('odd')
                        else:
                            if number == 8:
                                print('even')
                            else:
                                if number == 9:
                                    print('odd')
                                else:
```

Warmup Problem: Conditionals

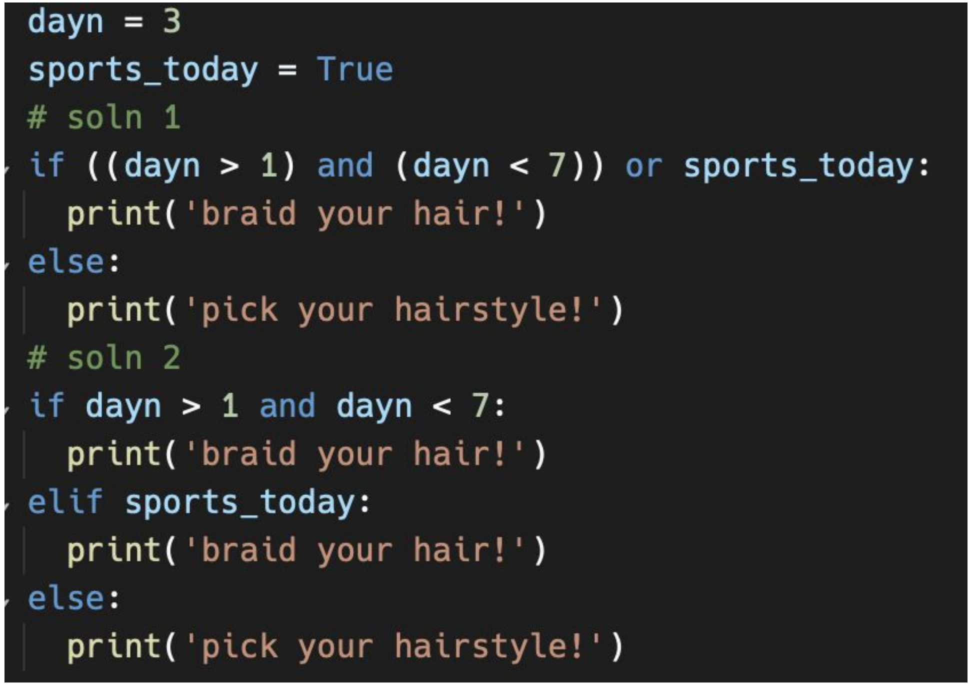
When you get ready in the morning for, you have to wear a braid to school (which happens Monday through Friday). On weekends (Saturday and Sunday), you wear a braid only when you have sports practice.
Write in a script which takes a number 1-7 to represent the day of the week (see below) and a boolean which tells you if you have sports practice. Print “BRAID” if you need to. Otherwise, print “NO BRAID”
*Optional challenge: Write this solution in **two** ways: one using an **or** statement, and one **without** an or statement.*

Day codes:

- 1 = sunday
- 2 = monday
- 3 = tuesday …
- 6 = friday
- 7 = sunday

Warmup Problem: Conditionals

When you get ready in the morning for, you have to wear a braid to school (which happens Monday through Friday). On weekends (Saturday and Sunday), you wear a braid only when you have sports practice.
Write in a script which takes a number 1-7 to represent the day of the week (see below) and a boolean which tells you if you have sports practice. Print “braid your hair!” if you need to. Otherwise, print “pick your hairstyle”
Write this solution in **two** ways: one using an **or** statement, and one **without** an or statement.



```python
dayn = 3
sports_today = True
# soln 1
if ((dayn > 1) and (dayn < 7)) or sports_today:
    print('braid your hair!')
else:
    print('pick your hairstyle!')
# soln 2
if dayn > 1 and dayn < 7:
    print('braid your hair!')
elif sports_today:
    print('braid your hair!')
else:
    print('pick your hairstyle!')
```

Lists – putting together multiple pieces of data!

- if we want to chain together multiple things in python, we can use square brackets to group them together [ ]
- a **list** is a **sequence** of **elements**
- we separate the **elements** by **commas**
- lists can contain any type of information
    - even other lists!
- empty brackets can denote an empty list we can add things to
    - think of the empty strings from yesterday!

[1, 4, 6, 4, 1]

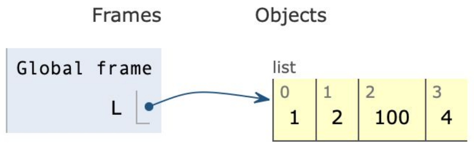
[1, 4, 6, 4, 1]

[1, 4, 6, 4, 1]

[1, True, 'hi', 2.5]

[1, [2], 1]

[] # empty list



Using lists: getting data

- storing data is fine, but how do we get (**retrieve**) the data??
- we also use square brackets for **indexing** (getting data)
- this is where counting from 0 becomes necessary and not just fun – we have to count the elements in our list starting with 0
    - second element is 1
    - third element is 2

```
      x = [1, 3, 3, 7]
index: 0  1  2  3
```

```
>>> x = [1, 3, 3, 7]
>>> x[0]
1
>>> x[1]
3
>>> x[3]
7
```

Using lists: Operators

len() returns the length of the list

in operator returns whether a list **contains** an element

- + will **concatenate** (stick together) two lists

```
>>> x = [1, 3, 3, 7]
>>> len(x)
4
>>> len([])
0
>>> 3 in x
True
>>> 9001 in x
False
>>> [1, 2] + [3]
[1, 2, 3]
```

Practice with lists

make a list containing the names of the people around you

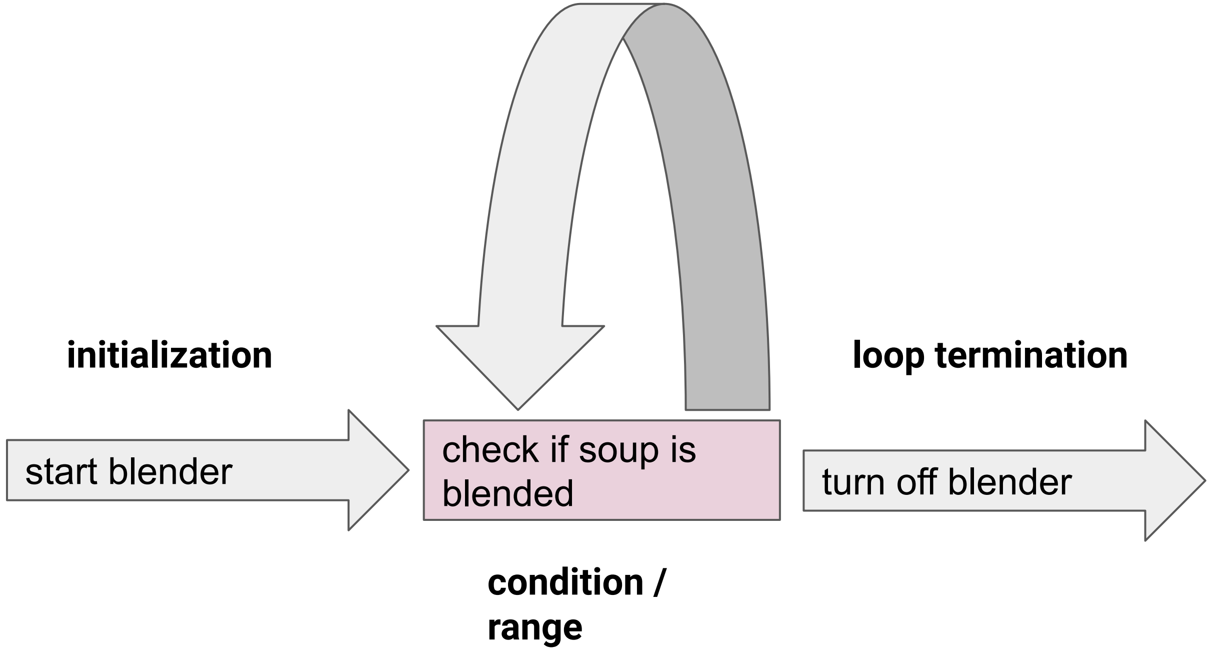
print out each name

challenge: print each name with an ! added, without adding the ! to the list

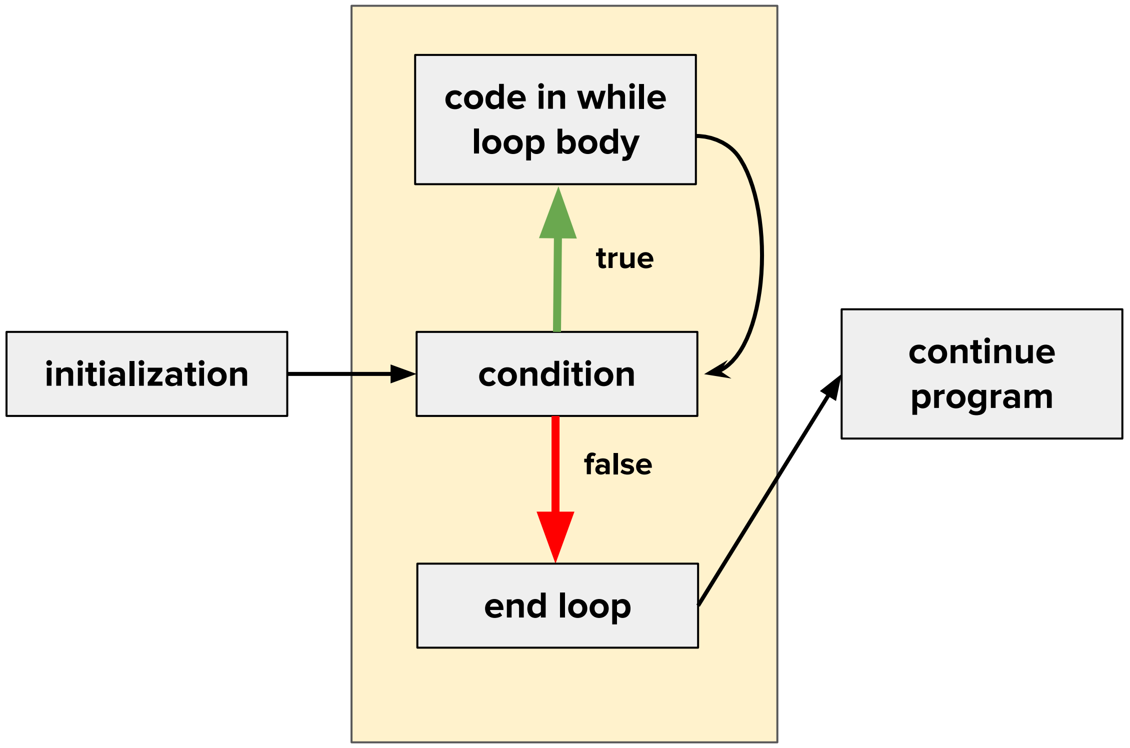
Yesterday

- reviewed **logic** (decide if something is true or false) and **conditionals** (use a true/false decision to run only selected code)
    - recipe metaphor: **vegetarian? if yes, add vegetable stock, if no, add chicken stock. *not both!***
- today: what if we want to run code more than once? can we do this flexibly?
    - blend the soup until no chunks remain
    - for every serving, chop 3 carrots
    - line every cupcake tin with paper

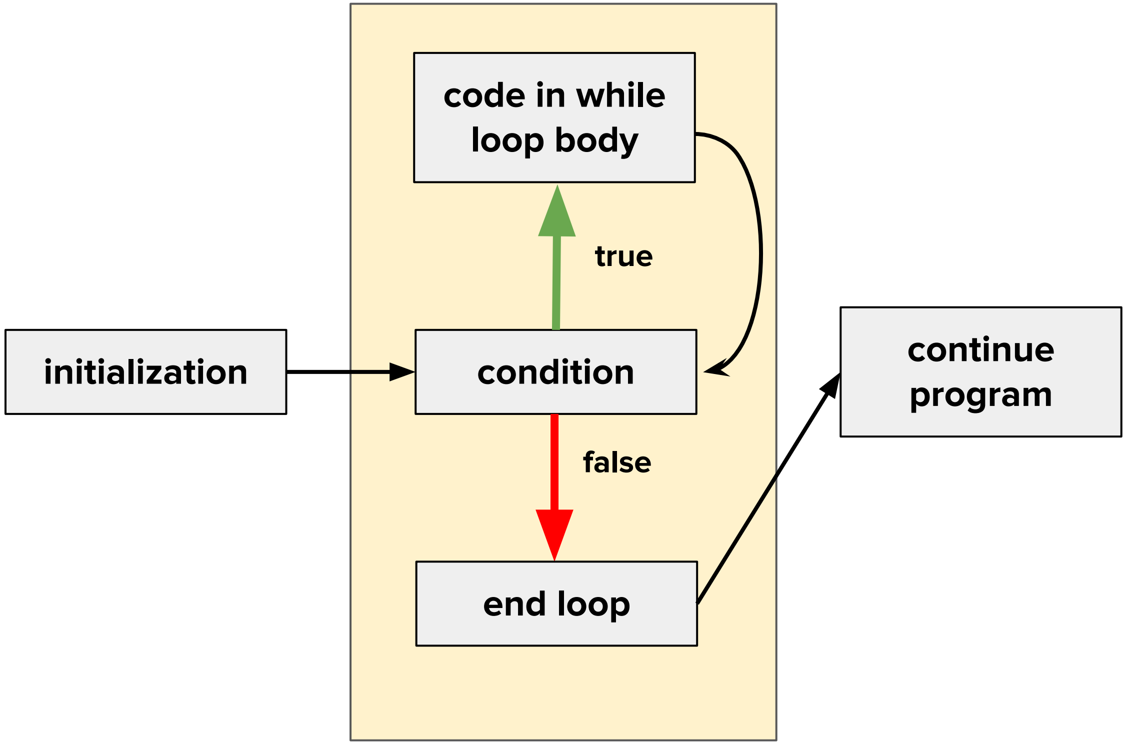
How does a loop work? Birds eye view and terminology



WHILE


code in while loop body
true
initialization
condition
continue program
false
end loop

WHILE



when to use:

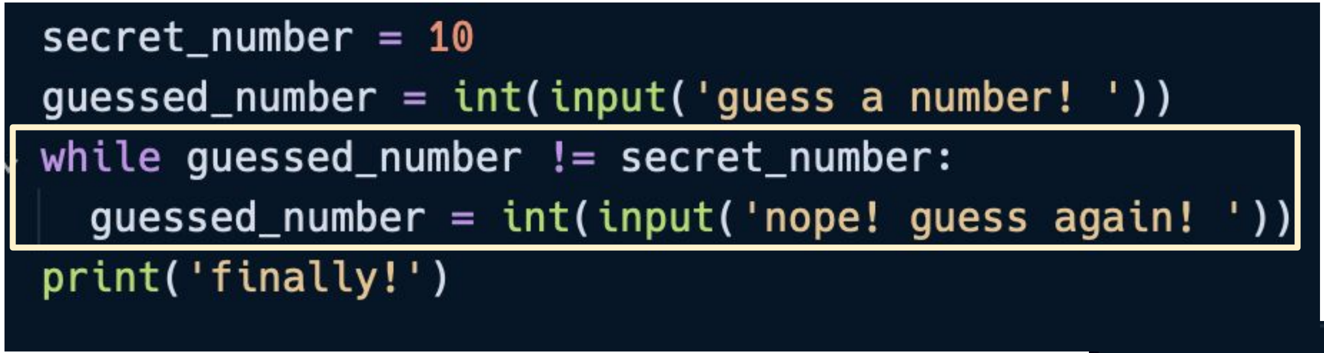
- you want to continue doing something UNTIL a condition changes, and you *don't know how long* that will take
- considered 'indefinite' iteration

examples:

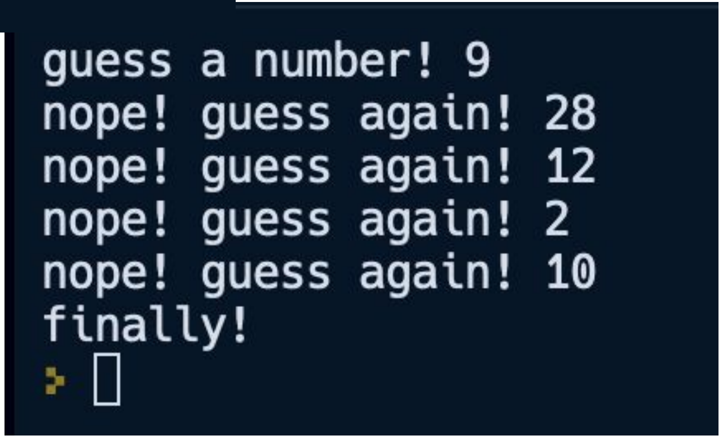
- collect coins up to a value
- braid hair
- while time hasn't run out, keep working

WHILE

indentation is still important!

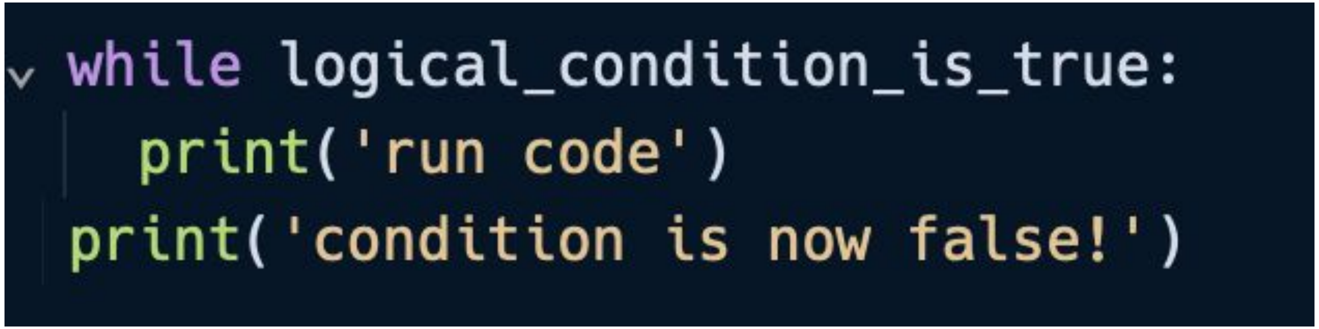


```python
secret_number = 10
guessed_number = int(input('guess a number! '))
while guessed_number != secret_number:
  guessed_number = int(input('nope! guess again! '))
print('finally!')
```

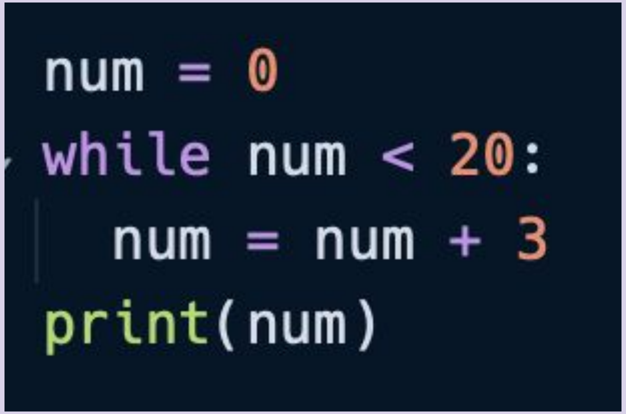


```
guess a number! 9
nope! guess again! 28
nope! guess again! 12
nope! guess again! 2
nope! guess again! 10
finally!
```

WHILE



```
while logical_condition_is_true:
    print('run code')
print('condition is now false!')
```



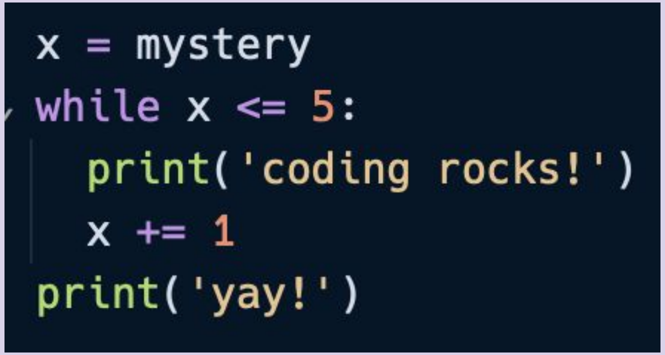
```
num = 0
while num < 20:
    num = num + 3
print(num)
```

how many times will the following code execute (how many times will it go inside the while loop)?

what will be the final value of num?

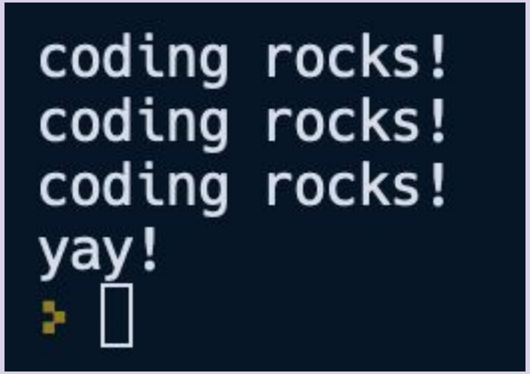
While Problem

you run this code



```
x = mystery
while x <= 5:
    print('coding rocks!')
    x += 1
print('yay!')
```

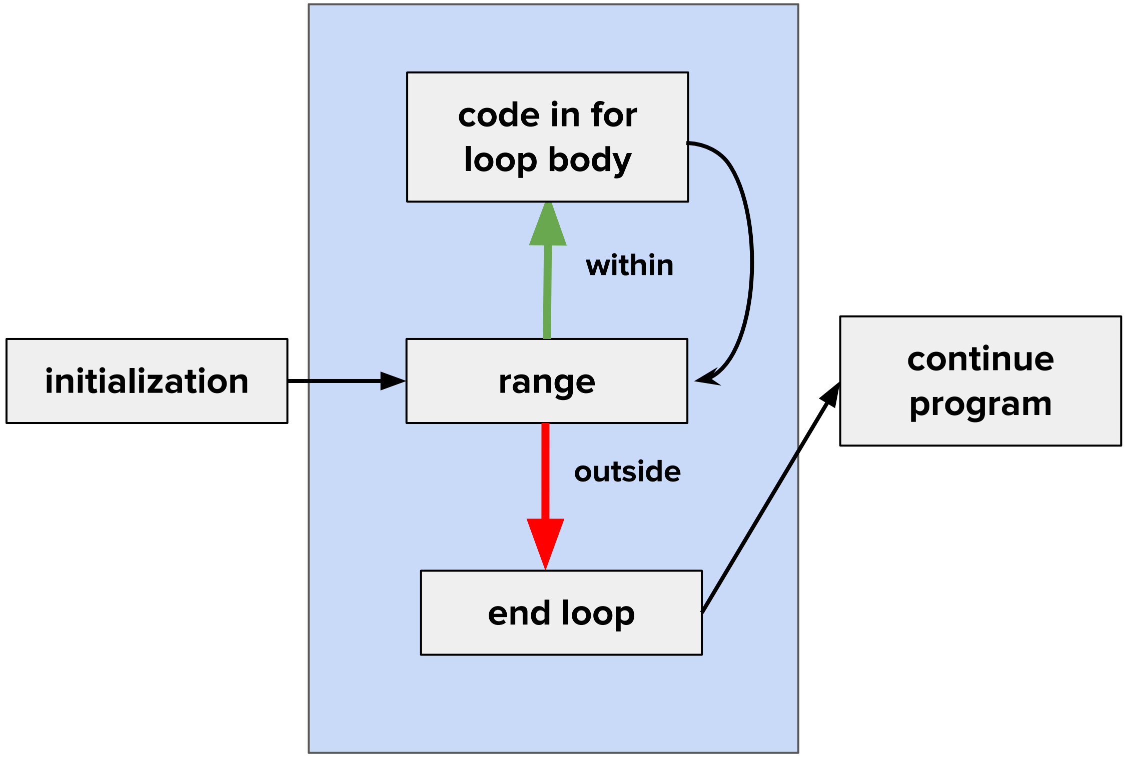
and see this output



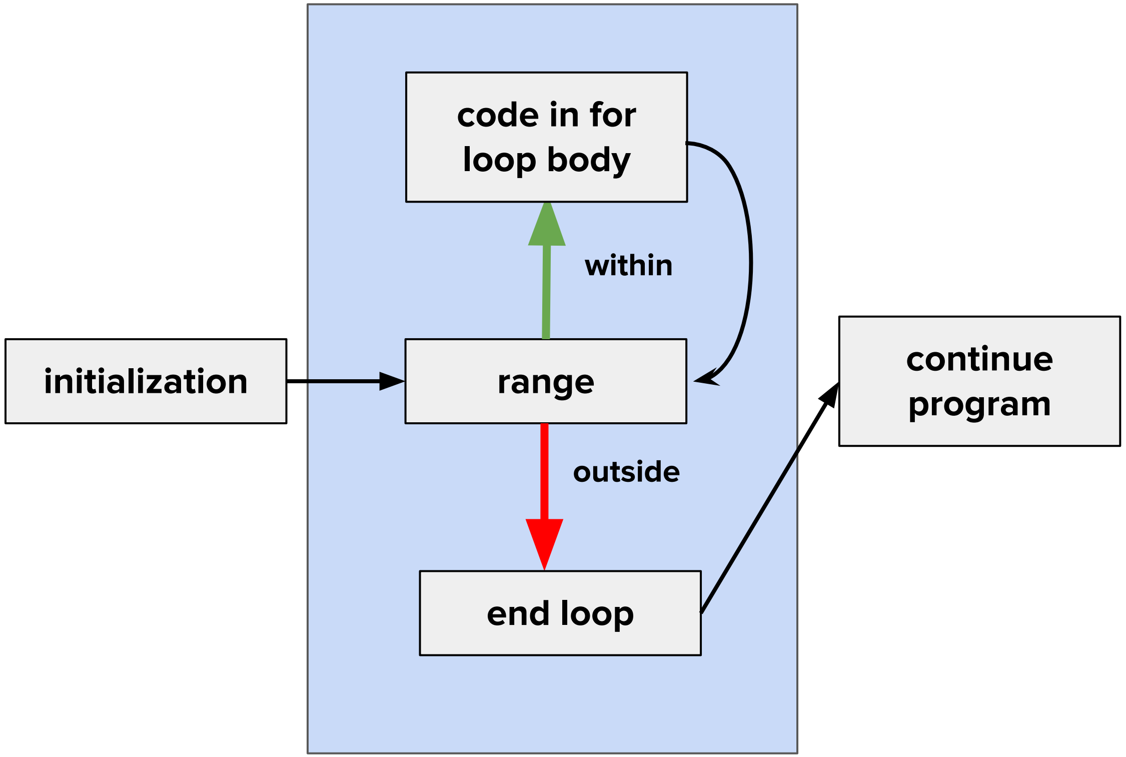
```
coding rocks!
coding rocks!
coding rocks!
yay!
>
```

what is the value of the variable mystery?

FOR


code in for loop body
within
initialization
range
outside
end loop
continue program

FOR



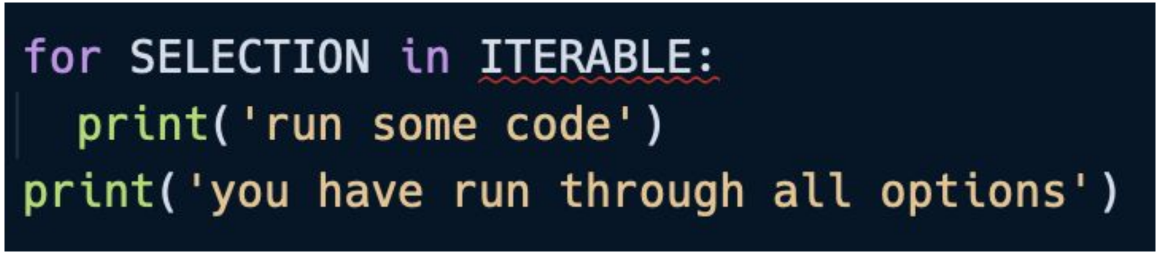
when to use:

- you want to do a process over a *KNOWN* set of information or repetitions
- considered 'definite' iteration

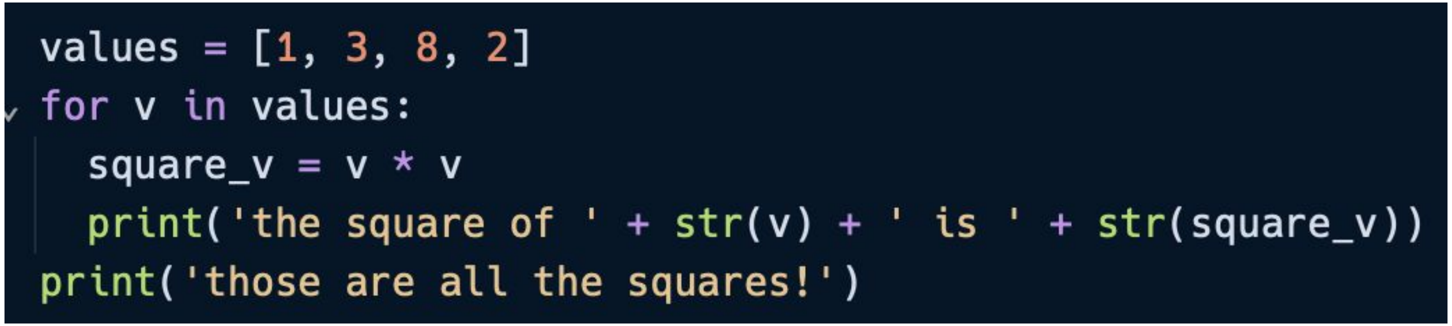
examples:

- put icing on each cupcake
- square a list of numbers
- check how many values in list exceed a certain number (think: points)
- push ups/cheer at football game

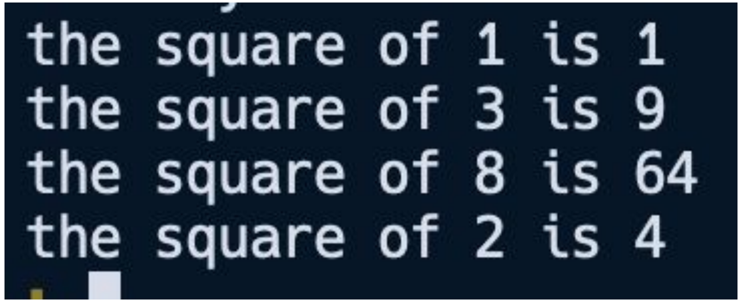
FOR



```python
for SELECTION in ITERABLE:
  print('run some code')
print('you have run through all options')
```



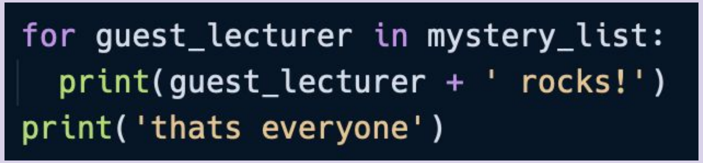
```python
values = [1, 3, 8, 2]
for v in values:
  square_v = v * v
  print('the square of ' + str(v) + ' is ' + str(square_v))
print('those are all the squares!')
```



```
the square of 1 is 1
the square of 3 is 9
the square of 8 is 64
the square of 2 is 4
```

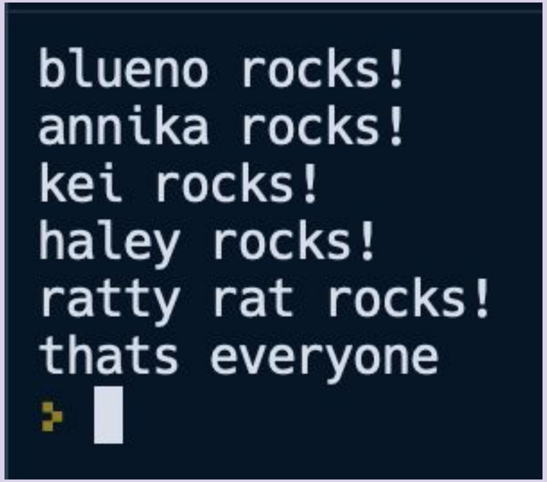
For Problem

you run this code



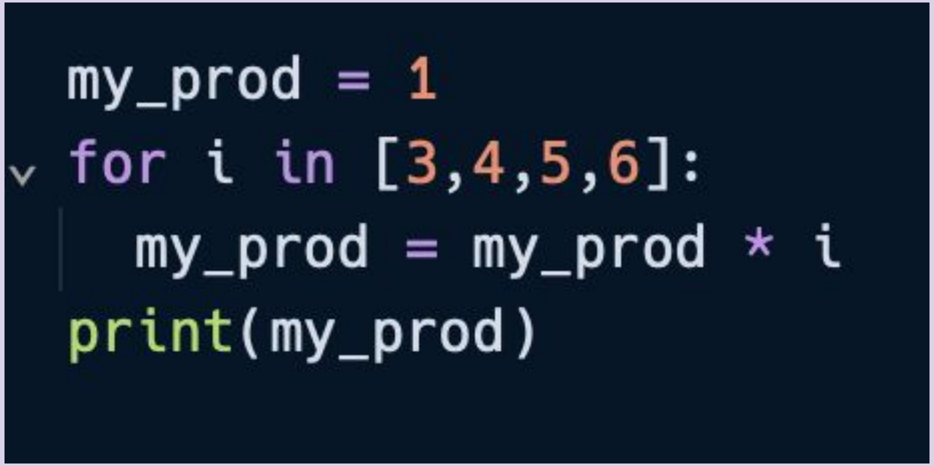
```python
for guest_lecturer in mystery_list:
  print(guest_lecturer + ' rocks!')
print('thats everyone')
```

and see this output



```
blueno rocks!
annika rocks!
kei rocks!
haley rocks!
ratty rat rocks!
thats everyone
> 
```

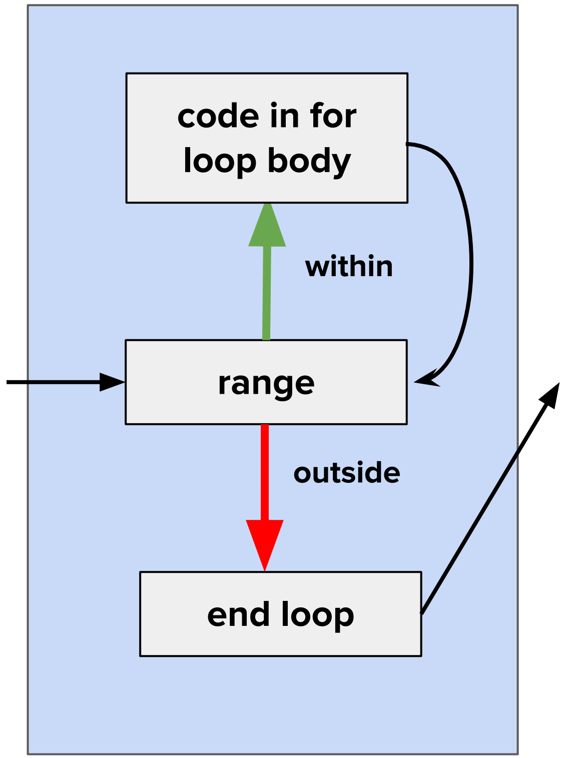
what is in the mystery_list, in order?



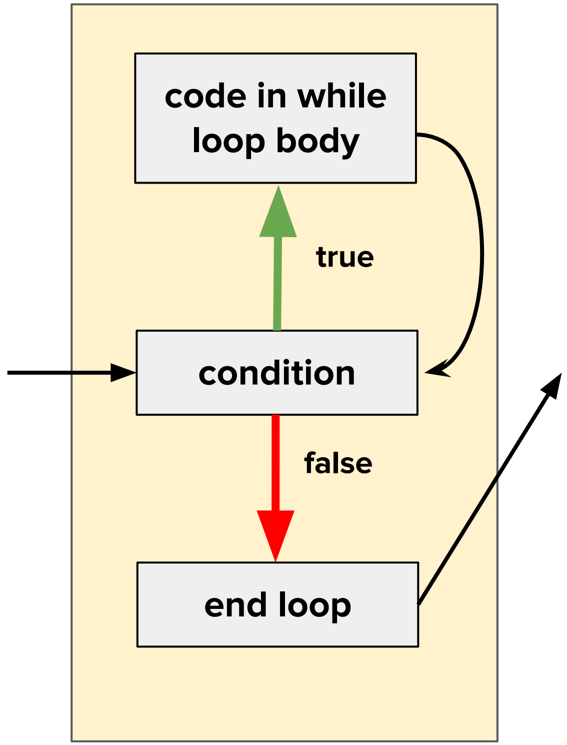
```python
my_prod = 1
for i in [3,4,5,6]:
    my_prod = my_prod * i
print(my_prod)
```

what was the result of this code?

FOR


code in for loop body
within
range
outside
end loop

WHILE


code in while loop body
true
condition
false
end loop

Think-Pair-Share

First: From the following list, can you identify which should be for loops and which should be while loops? *If you aren't sure, write down why you think it could be one or the other to discuss with your partner*

- you want to check how many of your kids is wearing a seatbelt
- you want to dig a hole to find buried treasure
- you want to visit all the dining halls on campus

If time: can you think of one new example (for code or from your life) that would fit in each type of loop?

Think-Pair-Share: Solution

First: From the following list, can you identify which should be for loops and which should be while loops? *If you aren't sure, write down why you think it could be one or the other to discuss with your partner*

- you want to check how many of your kids is wearing a seatbelt: **FOR**
- you want to dig a hole to find buried treasure**: WHILE**
- you want to visit all the dining halls on campus: **FOR**

class ideas:

- while I'm in metcalf, wear a jacket
- when washing dishes, for every dirty dish, wash
    - while dirty dishes, wash them
- while water bottle is not full, fill it with water
- when cleaning, while things on the floor pick up
- while reading, if not finished, keep reading
    - for each page, read it
- for every class, go to class

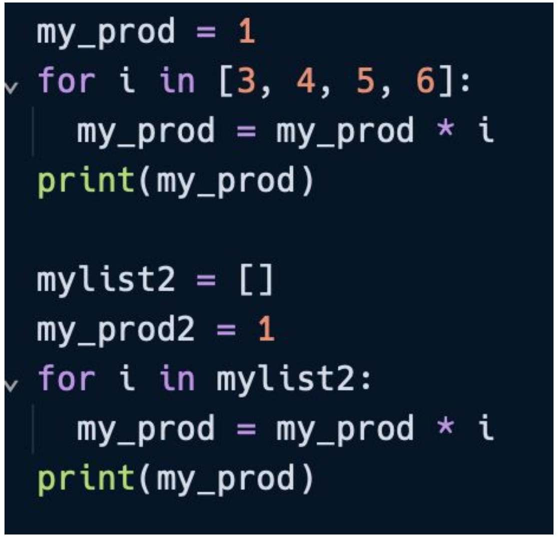
Some loops can be formatted both ways!

Common Bugs

never ending code (especially common with **while** loops)

never starting code (not always a bug!)

demos!



```
my_prod = 1
for i in [3, 4, 5, 6]:
    my_prod = my_prod * i
print(my_prod)

mylist2 = []
my_prod2 = 1
for i in mylist2:
    my_prod = my_prod * i
print(my_prod)
```

Practice problem 1: Adding to loops

- starting with the guessing game, instead you want to count the number of guesses it takes. add a variable that does this, and tell the user how many guesses it took them
    - start the variable outside – how many guesses is the MINIMUM?
    - how can you **increment** the variable (add value to it)?
    - where should this happen?
    - how can you share that with the user?

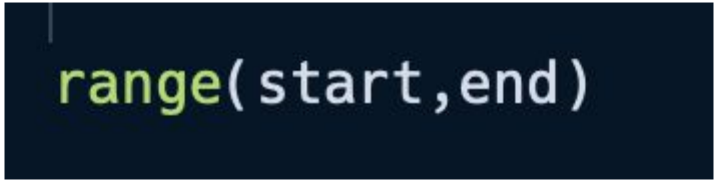
Practice problem 2: Changing loops

- now you want to limit the number of guesses the person can make to three. can you change the logic of the while loop to stop if they get to three guesses? add another boolean statement!

Range function – helps with for loops

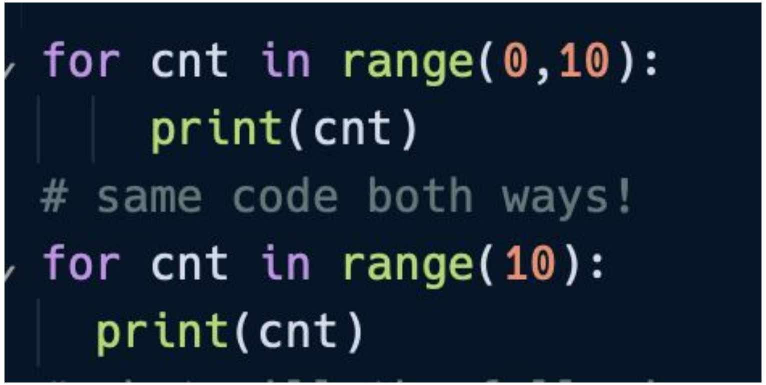
includes this number

not this number

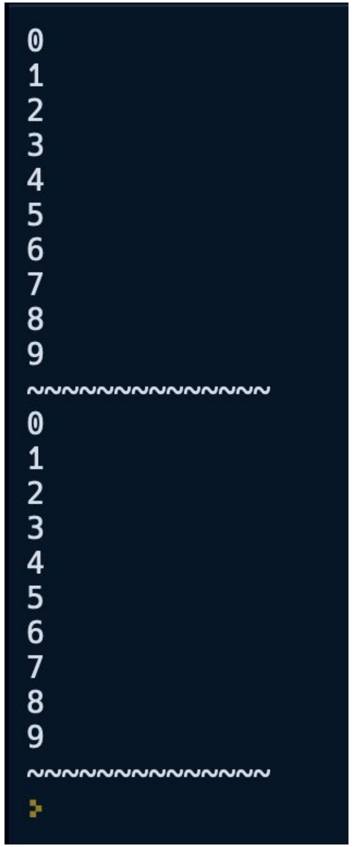


```
range(start,end)
```

if you only provide one number, python assumes you want the first number to be 0



```
for cnt in range(0,10):
    print(cnt)
# same code both ways!
for cnt in range(10):
  print(cnt)
```



```
0
1
2
3
4
5
6
7
8
9
~~~~~~~~~~~~~~
0
1
2
3
4
5
6
7
8
9
~~~~~~~~~~~~~~
>
```

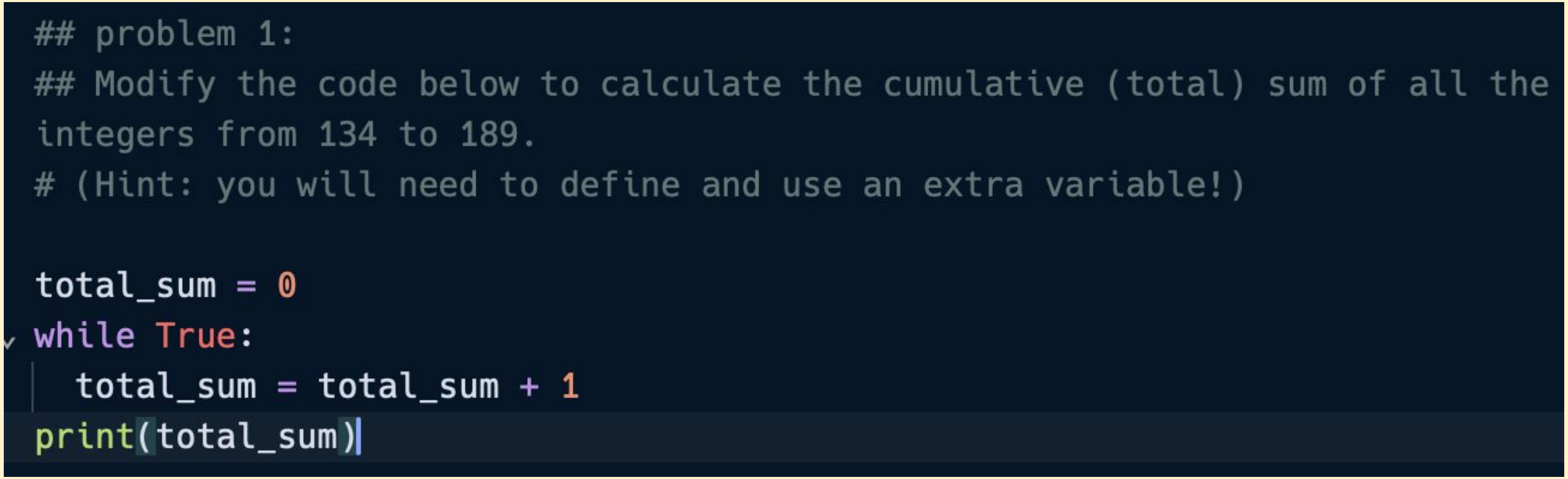
Practice Problem 3: For loops

go back to the code you wrote which prints the names of everyone in your list.

how could we use a for loop to accomplish this?

challenge: can you do it with and without the range function?

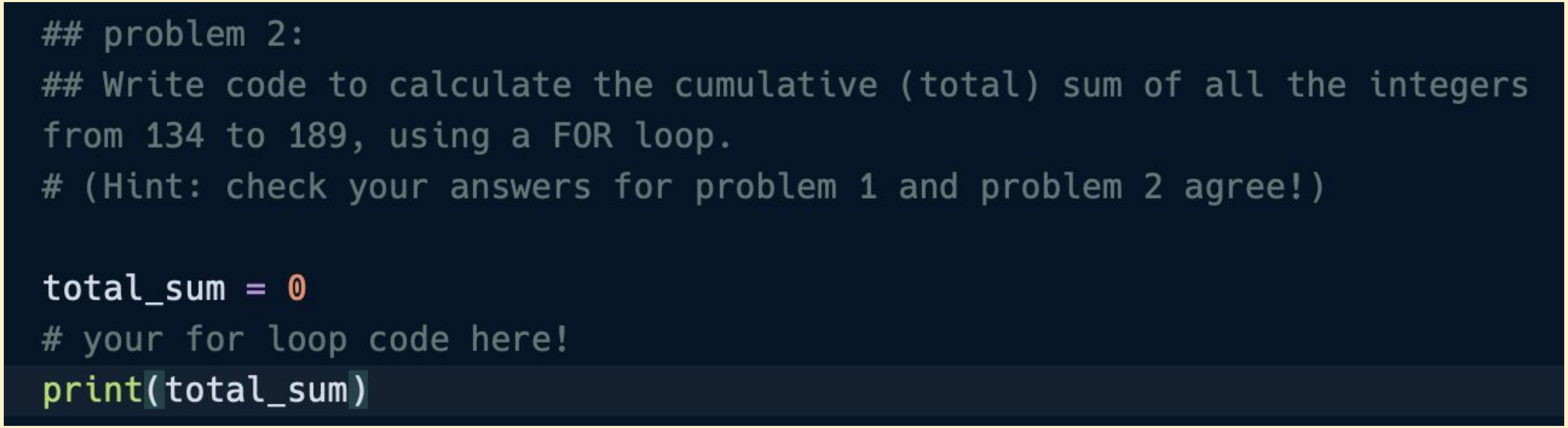
Homework Problem 1



```python
## problem 1:
## Modify the code below to calculate the cumulative (total) sum of all the integers from 134 to 189.
# (Hint: you will need to define and use an extra variable!)

total_sum = 0
while True:
    total_sum = total_sum + 1
print(total_sum)
```

Homework Problem 2



```
## problem 2:
## Write code to calculate the cumulative (total) sum of all the integers
from 134 to 189, using a FOR loop.
# (Hint: check your answers for problem 1 and problem 2 agree!)

total_sum = 0
# your for loop code here!
print(total_sum)
```

Homework problem 3

Update the guessing game code from class to tell the user if they are guessing too high or too low each time.

You should still print the number of guesses it took the user to find the number at the end of the code.

Homework problem 4

Write a script that calculates the sum of the following sequence

1 + ⅓ + ⅙ + 1/9 + 1/12 + …. + 1/999

put this total in the variable **total_sum** and print its value.

You can pick whether to use a for loop or while loop.

Optional challenge: write two versions of the code, with each type of loop!

Homework problem 5

The factorial of an integer, denoted as ! in math, is the product of that number and all of the positive integers less than it.

For example, 5! = 5 * 4 * 3 * 2 * 1 = 120.

Write a script to calculate N! when N is an integer. Use the command **while**. Store the result of your computation in the variable **fac_res**, which should be printed at the end of the code.

Python has a built-in function to calculate the factorial of an input but we ask you to **not** use it as we want you to write your own code.

`SyntaxError`

- **Cause**: code syntax mistake
- **Example**:

```
  File "file name", line number
    def incorrect(f)
                    ^
SyntaxError: invalid syntax
```

- **Solution**: the `^` symbol points to the code that contains invalid syntax. The error message doesn't tell you *what* is wrong, but it does tell you *where*.
- **Notes**: Python will check for `SyntaxErrors` before executing any code. This is different from other errors, which are only raised during runtime.

`IndentationError`

- **Cause**: improper indentation
- **Example**:

```
  File "file name", line number
    print('improper indentation')
IndentationError: unindent does not match any outer indentation level
```

- **Solution**: The line that is improperly indented is displayed. Simply re-indent it.
- **Notes**: If you are inconsistent with tabs and spaces, Python will raise one of these. Make sure you use spaces! (It's just less of a headache in general in Python to use spaces and all cs61a content uses spaces).

`TypeError`

- **Cause 1**:
  - Invalid operand types for primitive operators. You are probably trying to add/subract/multiply/divide incompatible types.
  - **Example**:

```
TypeError: unsupported operand type(s) for +: 'function' and 'int'
```

- **Cause 2**:
  - Using non-function objects in function calls.
  - **Example**:

```
>>> square = 3
>>> square(3)
Traceback (most recent call last):
  ...
TypeError: 'int' object is not callable
```

- **Cause 3**:
  - Passing an incorrect number of arguments to a function.
  - **Example**:

```
>>> add(3)
Traceback (most recent call last):
  ...
TypeError: add expected 2 arguments, got 1
```

`NameError`

- **Cause**: variable not assigned to anything OR it doesn't exist. This includes function names.
- **Example**:

```
File "file name", line number
  y = x + 3
NameError: global name 'x' is not defined
```

- **Solution**: Make sure you are initializing the variable (i.e. assigning the variable to a value) before you use it.
- **Notes**: The reason the error message says "global name" is because Python will start searching for the variable from a function's local frame. If the variable is not found there, Python will keep searching the parent frames until it reaches the global frame. If it still can't find the variable, Python raises the error.

`IndexError`

- **Cause**: trying to index a sequence (e.g. a tuple, list, string) with a number that exceeds the size of the sequence.
- **Example**:

```
File "file name", line number
  x[100]
IndexError: tuple index out of range
```

- **Solution**: Make sure the index is within the bounds of the sequence. If you're using a variable as an index (e.g. `seq[x]`, make sure the variable is assigned to a proper index.

Missing close quotes

This is similar to the previous bug, but much easier to catch. Python will actually tell you the line that is missing the quote:

```
File "file name", line "number"
  return 'hi
           ^
SyntaxError: EOL while scanning string literal
```

`EOL` stands for "End of Line."

`=` vs. `==`

The single equal sign `=` is used for *assignment*; the double equal sign `==` is used for testing equivalence. The most common error of this form is something like:

```
if x = 3:
```

Infinite Loops

Infinite loops are often caused by `while` loops whose conditions never change. For example:

```python
i = 0
while i < 10:
    print(i)
```

Sometimes you might have incremented the wrong counter:

```python
i, n = 0, 0
while i < 10:
    print(i)
    n += 1
```

Spelling

Python is *case sensitive*. The variable `hello` is not the same as `Hello` or `hello` or `helo`. This will usually show up as a `NameError`, but sometimes misspelled variables will actually have been defined. In that case, it can be difficult to find errors, and it is never gratifying to discover it's just a spelling mistake.

Missing Parentheses

A common bug is to leave off the closing parenthesis. This will show up as a `SyntaxError`. Consider the following code:

```
def fun():
    return foo(bar()   # missing a parenthesis here

fun()
```

Python will raise a `SyntaxError`, but will point to the line *after* the missing parenthesis:

```
File "file name", line "number"
    fun()
      ^
SyntaxError: invalid syntax
```

In general, if Python points a `SyntaxError` to a seemingly correct line, you are probably forgetting a parenthesis somewhere.