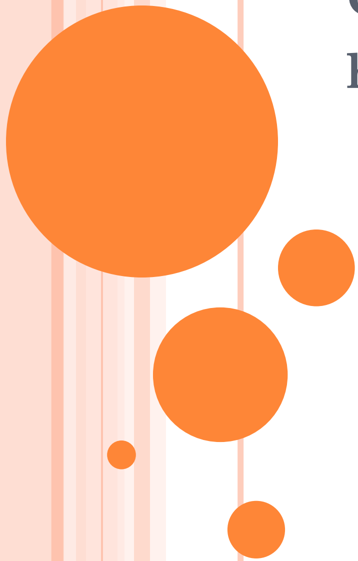
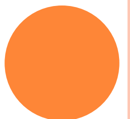
Python Repetition



Course Website:

https://www.cs.uoregon.edu/15U/cis122

Welcome Back! Time to get loopy…. Or slightly more than we have already ☺

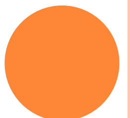


Square(): Revisited

Remember this:

```
import turtle

def line_and_turn(length, angle):
        turtle.fd(length)
        turtle.rt(angle)

def square(size):
        line_and_turn(size, 90)
        line_and_turn(size, 90)
        line_and_turn(size, 90)
        line_and_turn(size, 90)

def triangle(size):
        line_and_turn(size, 120)
        line_and_turn(size, 120)
        line_and_turn(size, 120)
```

we can dramatically improve this code with loops.

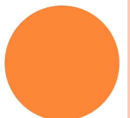


Why loops?

- Sometimes you need to run some lines of code multiple times. In particular maybe you don't know ahead of time (i.e. when *programming*) how many times the code must run, but you will know during *execution*.
- This is exactly what loops are for.
- Think about greeting guests for a party. You don't know how many people are coming but you need to greet each of them.

pseudocode:

```
while there are un-greeted guests:
greet the next guest
```



Loops

- two types of loops, **for** and **while**, we'll start by looking at **while** loops

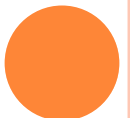
  syntax:

  while *conditional***:**

  [Code Block]

  code to run after the loop is done

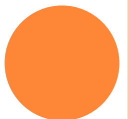
- The code will be executed over and over while the condition remains true. Once the condition is false the function will move on



Example of a loop

```
x = 0
while x <= 10:
        x += 1
        print(x)
```

- What will this do?
- What happens if we swap the lines of code.

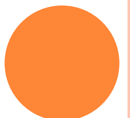


Time to get comfortable

- One downside to while loops is that accidental infinite loops are possible.

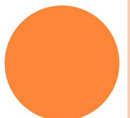
```
x = 0
while x <= 10:
        print(x)
```

- we left off the line that changed x, consequently the conditional never stops being true and the loop runs forever.
- ctrl-c to stop the execution (command-c on mac?)



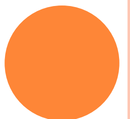
Square: re-revisited

```
import turtle

def line_and_turn(length, angle):
        turtle.fd(length)
        turtle.rt(angle)

def square(size):
        sides = 0
        while sides < 4:
                line_and_turn(size, 90)
                sides += 1

def triangle(size):
        sides = 0
        while sides < 3:
                line_and_turn(size, 120)
                sides += 1
```



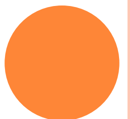
POLY()

- Let's try and see whether we can write a program that would generally define how to make a polygon of n- sides?

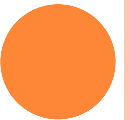


Poly()

```
import turtle

def line_and_turn(length, angle):
        turtle.fd(length)
        turtle.rt(angle)

def poly(size, num_sides):
        sides = 0
        while sides < num_sides:
                line_and_turn(size, 360/num_sides)
                sides += 1
```

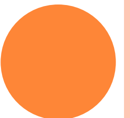


Welcome Back!



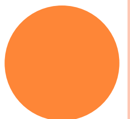
Indents

- As with functions and if statements indents are used to group text together, and once again mistaking what should be indented and what shouldn't is a VERY common error.



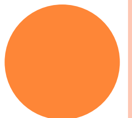
Spot the error

```
def loop_to_ten(x):
    while x < 10:
        x = 0
        print(x)
        x += 1
```



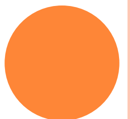
Spot The Error

```
def loop_to_ten(x):
    x = 0
    while x < 10:
        print(x)
        x += 1
```



Some Examples

- Loops lend themselves to some interesting problems, lets take a look at some



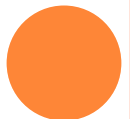
Compound interest

- Compound interest can be described thusly:

M(t+1) = M(t) + rM(t)

where M(t) is the money at time t
r is the interest rate (as a decimal)
and M(t+1) is the money after one unit of time (where the time unit is decided by how often the interest is compounded (yearly, daily, monthly)

- Lets write a function that takes a starting M, an amount of time, and a rate and figures out the final amount of money. We'll assume no withdrawals.



Fibonacci Sequence

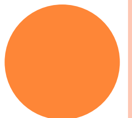
- let's make a function to print out all the Fibonacci numbers up to some arbitrary place.

- Fibonacci series:

  $f(1) = 1$

  $f(2) = 1$

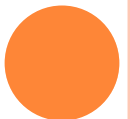
  $f(n) = f(n-2) + f(n-1)$



Checklist for loop writing

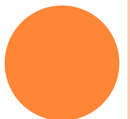
- Check list for while (indefinite) loops

1. Set up the loop/end condition
2. Initialize the loop variable (outside of the loop)
3. Write the body of the loop
4. Advance the counter variable
5. What to do when the loop is done?

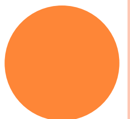


Any Questions?

- Then, Here are some of mine for you:
- Write a function to count up to 100 and print out all the even digits
- Write a function to print out all the numbers divisible by 3 between 3 and 30



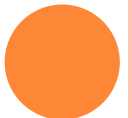
Welcome Back



Looping over sequences

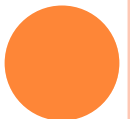
- Remember that strings are sequences of characters (strings are not the *only* sequences in python however).
- A very common activity is to loop over a sequence and do something with each item in that sequence.
- example-

count or print the individual characters in a string



Looping over sequences using while

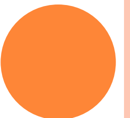
```
def count_letters(aString):
    index = 0
    count = 0

    while index < len(aString):
        char = aString[index]
        count += 1
        index += 1
    return count
```



For Loop

- For loops are specifically designed to run over a sequence applying the code within the loop once for each object in the sequence.

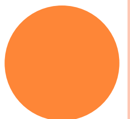
- This specialization makes them less flexible but also a bit safer and easier to use.



Looping comparisons for looping over strings

```
def count_lowercase(aString):
    index = 0
    count = 0

    while index < len(aString):
        char = aString[index]
        if char.islower():
            count += 1
        index += 1
    return count
```

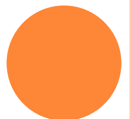
```
def count_lowercase2(aString):
    count = 0

    for char in aString:
        if char.islower():
            count += 1
    return count
```



For Loop Syntax

```
for <obj> in <seq>:
        <code to execute>

for char in aString:
        print(char)

for i in str1:
        print(i)
```

- Note both code snippets above work *exactly* the same. The differences in the names (char/i and aString/str1) are purely cosmetic. They are variables standing in for the actual data we will have later.



Advantages and Disadvantages

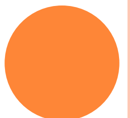
- Advantages-

Because the for loop operates once for each item in the sequence it is impossible to accidentally cause an infinite loop.

The for loop is more compact and sleek when operating over a sequence than the while because it is specially built for that task.

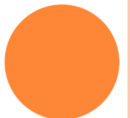
- Disadvantages-

The while loop is much more flexible. If you want to get every other item from a sequence you can do it with a for loop but it's probably just as easy to use a while loop (actually you can do this specific thing with a slice very easily).



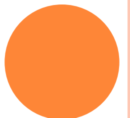
Continue and break statements

- continue and break are keywords that allow us to control the loop.
- The continue statement allows us to continue running the loop while ignoring the rest of the code within the loop for that particular iteration.
- break on the other hand allows us stop the execution of a loop midway and continue with the rest of the code.

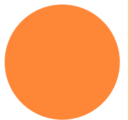


Check strings for numbers

- Lets write a function to check each character of a string and if the character is a digit (i.e. number) it prints out that digit. It also keeps track of the number of digits found and prints that number at the end.
- Don't try to build it all at once, do one piece at a time. Let's start by building a function that prints out each letter in a string in turn.
- Here slicing really shines.

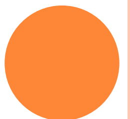


Welcome Back



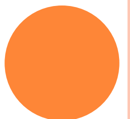
Loops w/o Sequences

- If you just want a loop to run a certain number of times there's a way to do that with a for loop.
- All of our examples yesterday involved manipulating a sequence as part of the loop, but maybe we just want the sequence to serve as a counter of sorts.



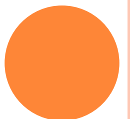
RANGE()

- the built-in range() function gives us a sequence of numbers from 0 up to the argument value passed to range.
- range(5) for instance gives us a sequence that runs from 0 to 4.
- range() can also be called with 1, 2 or 3 arguments.
  - range(1,5) gives us a sequence from 1 to 4.
  - range(0, 5, 2) gives us a sequence from 0 to 4 by twos (i.e. 0, 2, 4)
  - Range(starting value, ending value, interval)



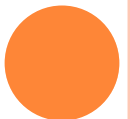
Default values

- range() can take 1, 2 or 3 inputs.



COUNT_TO()

- let's write a function that counts (prints) the numbers up to some argument we give the function.
- Let's write it both as a while loop and a for loop.

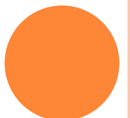


NESTED LOOPS

- Aka loopception!

As I was going to St. Ives,
I met a man with seven wives,
Each wife had seven sacks,
Each sack had seven cats,
Each cat had seven kits:
Kits, cats, sacks, and wives,
How many were there going to St. Ives?

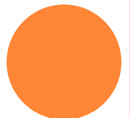
- Of course the poem above is a trick question but let's say we wanted to calculate the number of kits, cats, sacks, and wives. We could use algebra, but let's make the computer do the work for us.



Pseudo-code

```
def st_ives():
    for each wife
        count the wife
        for each sack
            count the sack
            for each cat
                count the cat
                for each kitten
                    count the kitten
    announce the total
```

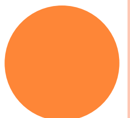
Okay let's write the function



Looping without loops

- You remember me mentioning something about recursion?
- Recursion is a way of looping without actually using loops. We aren't going to have time to really learn recursion but I want you to get a taste of it.

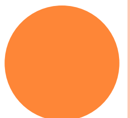
```
def countdown(n):
 if n == 0:
  print('Blastoff!')
 else:
  print(n)
  countdown(n - 1)

countdown(5)
```



Recursion basics

- Connection to Mathematical induction.
- The essence of recursion is to have a base case, below the base case is when n==0, and if you aren't at the base case the function calls itself on an input that is closer to the base case.

```
def countdown(n):
 if n == 0:
  print('Blastoff!')
 else:
  print(n)
  countdown(n - 1)

countdown(5)
```



QUESTION TIME

- We did the counting, now how about implementing our even-odd printer using a for loop with a sequence.
- Can we use range() with other sequences?
- Implement a function to ask the user for an input and print that input until the person enters 'quit'.