

# CompSci 101

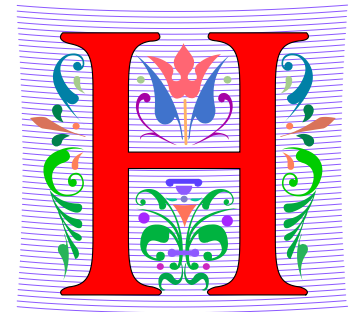
## Turtles, for loop, accumulation, range

Susan Rodger

February 9, 2023

create a new empty string **ret**  
for each character **ch** in the **phrase**  
    if **ch** is not a vowel  
        add **ch** to the end of **ret**  
**ret** is the result

# H is for ...



- **HTTP**
  - A Protocol we use every day, and HTTPS
- **Hello World**
  - The quintessential first program: 40 years ago!  
<http://helloworldcollection.de/>
- **Hack**
  - Hacker, Hacktivism, Hack Duke
- **Hashing**
  - How Dictionaries work

# Brian Fox



- See Wikipedia: <http://bit.ly/brianfox2018>
  - Bash Shell, Stallman, Wells Fargo, more
- See LifeHacker: <http://bit.ly/brianfox-hack>
  - Learned Logo at 8, wrote it at 21 for Apple!
  - Open Voting

***There's nothing that I am better at than everyone else, except being me. There's no secret to being me. Follow your interests and work hard at them. Then you will play bass better, program better, cook better, ride motorcycles better, or anything else that you really want to do.***

# Announcements

- **Assignment 2 out – Quiz due 2/14, program 2/16**
- **APT-2 due tonight!**
- **Lab 4 Friday**
  - Complete prelab before going to lab
- **Reading and Sakai quizzes for next week up today**
- **Exam 1**
  - Do not discuss until it is handed back!

# PFTD

- **Import a file**
- **String and List Functions**
- **Turtles**
- **For loop/Accumulation**

Main: `if __name__ == '__main__':`

- Main – where Python starts and ends in some file
- Consider file `Food.py`

```
def makeSandwich(food):  
    print("Making the sandwich", food)  
  
if __name__ == '__main__':  
    makeSandwich('peanut butter and jelly')
```

Main: `if __name__ == '__main__':`

- Main – where Python starts and ends in some file
- Consider file `Food.py`

```
def makeSandwich(food):  
    print("Making the sandwich", food)
```

```
if __name__ == '__main__':  
    makeSandwich('peanut butter and jelly')
```

Execution  
starts here:

OUTPUT:

Making the sandwich peanut butter and jelly

# Main vs. Import

- **Import – another file with useful code (functions)**

- Ignores `if __name__ == '__main__':` in the other file

- **Food.py**

```
def makeSandwich(food):  
    print("Making the sandwich", food)  
  
if __name__ == '__main__':  
    makeSandwich('peanut butter and jelly')
```

- **FoodFriend.py**

```
import Food  
  
if __name__ == '__main__':  
    Food.makeSandwich("bacon, lettuce, and tomato")
```



# Main vs. Import

- **Import – another file with useful code (functions)**

- Ignores `if __name__ == '__main__':` in the other file

- **Food.py**

```
def makeSandwich(food):  
    print("Making the sandwich", food)  
  
if __name__ == '__main__':  
    makeSandwich('peanut butter and jelly')
```

- **FoodFriend.py**

Food 'dot' tells you which file `makeSandwich` function is in

```
import Food  
  
if __name__ == '__main__':  
    Food.makeSandwich("bacon, lettuce, and tomato")
```

Food is file where function `makeSandwich` is

# Run FoodFriend.py

- **FoodFriend.py**

```
import Food

if __name__ == '__main__':
    Food.makeSandwich("bacon, lettuce, and tomato")
```

# Run FoodFriend.py

- **FoodFriend.py**

Grab functions from Food.py



```
import Food

if __name__ == '__main__':
    Food.makeSandwich("bacon, lettuce, and tomato")
```

# Run FoodFriend.py

- **FoodFriend.py**



```
import Food

if __name__ == '__main__':
    Food.makeSandwich("bacon, lettuce, and tomato")
```

Here is function makeSandwich

- **Food.py**

```
def makeSandwich(food):
    print("Making the sandwich", food)
```

Ignores code  
in main


```
if __name__ == '__main__':
    makeSandwich('peanut butter and jelly')
```

# Run FoodFriend.py

- **FoodFriend.py**

```
import Food

if __name__ == '__main__':
    Food.makeSandwich("bacon, lettuce, and tomato")
```



Use the makeSandwich function in the file Food.py

- **Food.py**

```
def makeSandwich(food):
    print("Making the sandwich", food)

if __name__ == '__main__':
    makeSandwich('peanut butter and jelly')
```

# Run FoodFriend.py

- **FoodFriend.py**

```
import Food

if __name__ == '__main__':
    Food.makeSandwich("bacon, lettuce, and tomato")
```



OUTPUT:

Making the sandwich bacon, lettuce and tomato

- **Food.py**

```
def makeSandwich(food):
    print("Making the sandwich", food)

if __name__ == '__main__':
    makeSandwich('peanut butter and jelly')
```

# Main vs. Import

- **Run main in Food.py**
  
  
  
  
  
  
  
  
  
  
- **Run main in FoodFriend.py**

# Main vs. Import

- **Run main in Food.py**

```
Making the sandwich peanut butter and jelly
```

- **Run main in FoodFriend.py**

```
Making the sandwich bacon, lettuce, and tomato
```



# Food2.py – Bad Code!

- Food2.py – Modification of Food.py

```
def makeSandwich(food):  
    print("Making the sandwich", food)  
  
→ makeSandwich('hummus and sprouts')  
  
if __name__ == '__main__':  
    makeSandwich('peanut butter and jelly')
```

# Food2.py – Bad Code!

- Food2.py – Modification of Food.py

```
def makeSandwich(food):  
    print("Making the sandwich", food)  
makeSandwich('hummus and sprouts')  
  
if __name__ == '__main__':  
    makeSandwich('peanut butter and jelly')
```

Code not indented, this is bad code!!!!

# Modify Main

- **Food2.py**

```
def makeSandwich(food):  
    print("Making the sandwich", food)  
  
makeSandwich('hummus and sprouts')  
  
if __name__ == '__main__':  
    makeSandwich('peanut butter and jelly')
```

Line of code will execute when this file is imported. Then you are executing code more than just the functions in that file.

Code not indented, this is bad code!!!!

# Modified Main vs. Import

- **Run main in Food2.py**

```
Making the sandwich hummus and sprouts  
Making the sandwich peanut butter and jelly
```

- **Run main in FoodFriend.py with import of Food2.py**

```
Making the sandwich hummus and sprouts  
Making the sandwich bacon, lettuce, and tomato
```

Remember, ALWAYS put code in main or in a function, indented!

# More String and List Methods

## String

<code>.find(s)</code>	index of first occurrence of s
<code>.rfind(s)</code>	index of last occurrence of s (from Right)
<code>.upper()/ .lower()</code>	uppercase/lowercase version of string
<code>.strip()</code>	remove leading/trailing whitespace
<code>.count(s)</code>	number of times see s in string
<code>.startswith(s)</code>	bool of whether the string begins with s
<code>.endswith(s)</code>	bool of whether the string ends with s

## List

<code>sum(lst)</code>	sum of the elements in lst
<code>max(lst)</code>	maximum value of lst
<code>min(lst)</code>	minimum value of lst
<code>.append(elm)</code>	Mutates the list by adding elm to the end of the list
<code>.count(elm)</code>	Number of times see elm in the list

# Some String Methods

```
str = 'ghosts'
```

```
x = str.find('s')
```

```
x = str.rfind('s')
```

```
x = str.upper()
```

```
x = str.count('s')
```

```
x = str.startswith('gh')
```

```
x = str.endswith('s')
```

```
x = str.endswith('t')
```

# Some String Methods

```
str = 'ghosts'
```

```
x = str.find('s')
```

```
x = str.rfind('s')
```

```
x = str.upper()
```

```
x = str.count('s')
```

```
x = str.startswith('gh')
```

```
x = str.endswith('s')
```

```
x = str.endswith('t')
```

```
str is 'ghosts'
```

```
x is 3
```

```
x is 5
```

```
x is 'GHOSTS'
```

```
x is 2
```

```
x is True
```

```
x is True
```

```
x is False
```

# Some List Methods

```
lst = [5, 2, 4, 5, 5]
```

```
lst is [5, 2, 4, 5, 5]
```

```
x = max(lst)
```

```
x = min(lst)
```

```
x = sum(lst)
```

```
x = lst.count(5)
```

```
lst.append(7)
```

```
x = lst.append(3)
```



# Some List Methods

```
lst = [5, 2, 4, 5, 5]
```

```
x = max(lst)
```

```
x = min(lst)
```

```
x = sum(lst)
```

```
x = lst.count(5)
```

```
lst.append(7)
```

```
x = lst.append(3)
```

Correct way to use append

Don't use append this way, no return value

```
lst is [5, 2, 4, 5, 5]
```

```
x is 5
```

```
x is 2
```

```
x is 21
```

```
x is 3
```

```
lst is [5, 2, 4, 5, 5, 7]
```

```
x is None and lst is [5, 2, 4, 5, 5, 7, 3]
```

# Three ways to use functions with List

- `x = max(lst)`
  - `lst` is parameter to the `max` function that has a return value
- `x = lst.count(5)`
  - `lst` has its own functions that can be applied to a list
- `lst.append(7)`
  - mutates/changes `lst`, no return value

```
m sort(self, key=None, reverse=False)
m pop(self, index)
m copy(self)
m count(self, value)
m index(self, value, start, stop)
m append(self, object)
m clear(self)
m extend(self, iterable)
m insert(self, index, object)
m remove(self, value)
m reverse(self)
m add(self, value)
```

Press Ctrl+. to choose the selected (or first) suggestion and insert a dot afterwards [Next Tip](#)

`lst.`

# Three ways to use functions with List

- `x = max(lst)`
  - `lst` is parameter to the `max` function that has a return value
- `x = lst.count(5)`
  - `lst` has its own functions that can be applied to a list
- `lst.append(7)`
  - mutates/changes `lst`, no return value

```
m sort(self, key=None, reverse=False)
m pop(self, index)
m copy(self)
m count(self, value)
m index(self, value, start, stop)
m append(self, object)
m clear(self)
m extend(self, iterable)
m insert(self, index, object)
m remove(self, value)
m reverse(self)
m add (self, value)
```

Press Ctrl+. to choose the selected (or first) suggestion and insert a dot afterwards [Next Tip](#)

`lst.`

When you type the name of a list followed by `.` then PyCharm shows you all the list functions

# Three ways to use functions with List

- `x = max(lst)`

Use list as parameter

- `lst` is parameter to the `max` function that has a return value

- `x = lst.count(5)`

Use list function with a return value

- `lst` has its own functions that can be applied to a list

- `lst.append(7)`

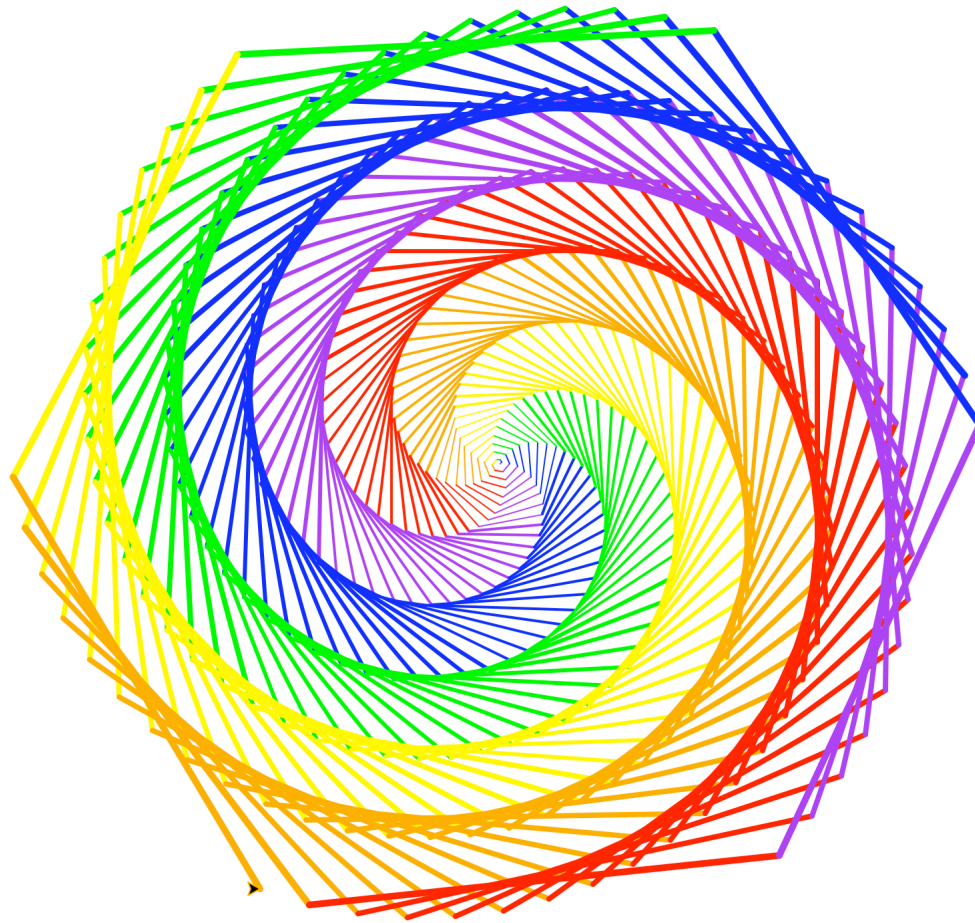
Use list function that mutates, no return value

- mutates/changes `lst`, no return value

# WOTO-1 – Import, Strings and Lists

<http://bit.ly/10123s-0209-1>

# Run Turtle, Run



# Turtle Programming

- **Must:**
  - Import turtle module
  - Create window/Screen
  - Last thing - exit on click
  - Create turtles to use, name/type/value
- **Review Turtle commands and concepts**
  - [http://bit.ly/turtle\\_tutorial](http://bit.ly/turtle_tutorial) for more, and book
- **See ColorMyWorld.py, and Spiro.py for some ideas**
  - Color, Position, Leaving Turtle where started
  - Many more commands than this



# Put yourself in the turtle t...

```
t.forward(50)           # turtle moves forward  
                        # drawing a line  
t.left(90)             # turtle turns to its left  
t.pencolor("blue")     # change pen color  
t.forward(100)         # turtle moves forward  
                        # drawing line, new color
```

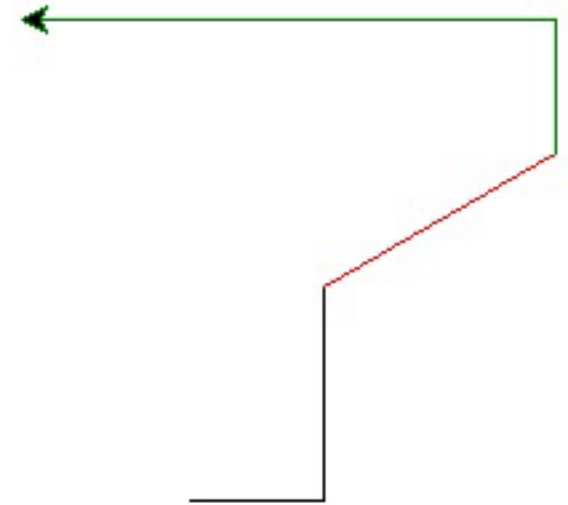


# Example: simple.py

```
import turtle

def drawPicture(turt):
    t.forward(50)
    t.left(90)
    t.forward(80)
    t.pencolor('red')
    t.right(60)
    t.forward(100)
    t.pencolor('green')
    t.left(60)
    t.forward(50)
    t.left(90)
    t.forward(200)

if __name__ == '__main__':
    win = turtle.Screen()
    t = turtle.Turtle()
    drawPicture(t)
    win.exitonclick()
```



# Example: Simple.py parts

```
import turtle ←
```

- Import at the top

```
]if __name__ == '__main__':  
    win = turtle.Screen()  
    t = turtle.Turtle()  
    drawPicture(t)  
]    win.exitonclick()
```

---

# Example: Simple.py parts

```
import turtle
```

```
]if __name__ == '__main__':
```

```
→ win = turtle.Screen()
```

```
    t = turtle.Turtle()
```

```
    drawPicture(t)
```

```
]    win.exitonclick()
```

- **Create window**

# Example: Simple.py parts

```
import turtle
```

```
]if __name__ == '__main__':  
    win = turtle.Screen()  
    → t = turtle.Turtle()  
    drawPicture(t)  
]    win.exitonclick()
```

- Create turtle with name t

# Example: Simple.py parts

```
import turtle
```

```
if __name__ == '__main__':  
    win = turtle.Screen()  
    t = turtle.Turtle()  
    drawPicture(t)  
    win.exitonclick()
```

- **Call function to draw and pass the turtle t as an argument**

# Example: Simple.py parts

```
import turtle
```

```
if __name__ == '__main__':  
    win = turtle.Screen()  
    t = turtle.Turtle()  
    drawPicture(t)  
    win.exitonclick()
```

- Close canvas when click on it

# Example: Simple.py DrawPicture

```
def drawPicture(turt):  
    turt.forward(50)  
    turt.left(90)  
    turt.forward(80)  
    turt.pencolor('red')  
    turt.right(60)  
    turt.forward(100)  
    turt.pencolor('green')  
    turt.left(60)  
    turt.forward(50)  
    turt.left(90)  
    turt.forward(200)
```



# Example: Simple.py DrawPicture

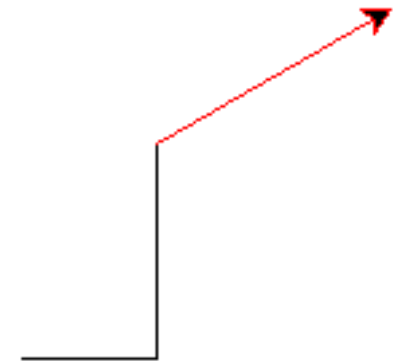
```
def drawPicture(turt):  
    turt.forward(50)  
    turt.left(90)  
    turt.forward(80)  
    turt.pencolor('red')  
    turt.right(60)  
    turt.forward(100)  
    turt.pencolor('green')  
    turt.left(60)  
    turt.forward(50)  
    turt.left(90)  
    turt.forward(200)
```





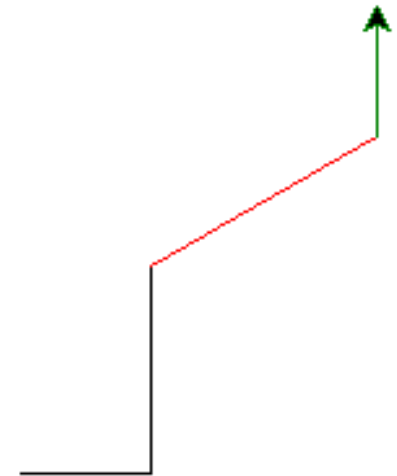
# Example: Simple.py DrawPicture

```
def drawPicture(turt):  
    turt.forward(50)  
    turt.left(90)  
    turt.forward(80)  
    turt.pencolor('red')  
    turt.right(60)  
    turt.forward(100)  
    turt.pencolor('green')  
    turt.left(60)  
    turt.forward(50)  
    turt.left(90)  
    turt.forward(200)
```



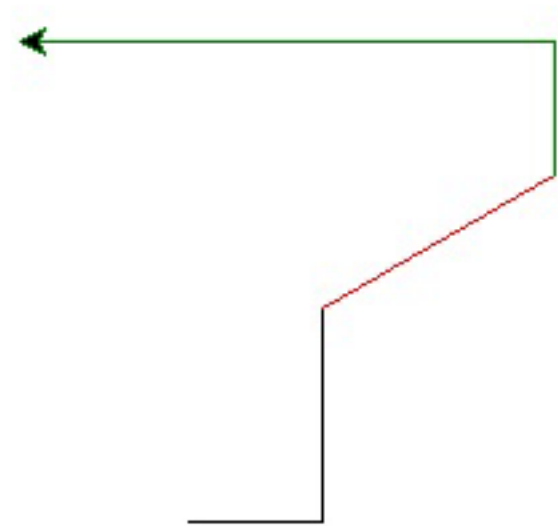
# Example: Simple.py DrawPicture

```
def drawPicture(turt):  
    turt.forward(50)  
    turt.left(90)  
    turt.forward(80)  
    turt.pencolor('red')  
    turt.right(60)  
    turt.forward(100)  
    turt.pencolor('green')  
    turt.left(60)  
    turt.forward(50)  
    turt.left(90)  
    turt.forward(200)
```

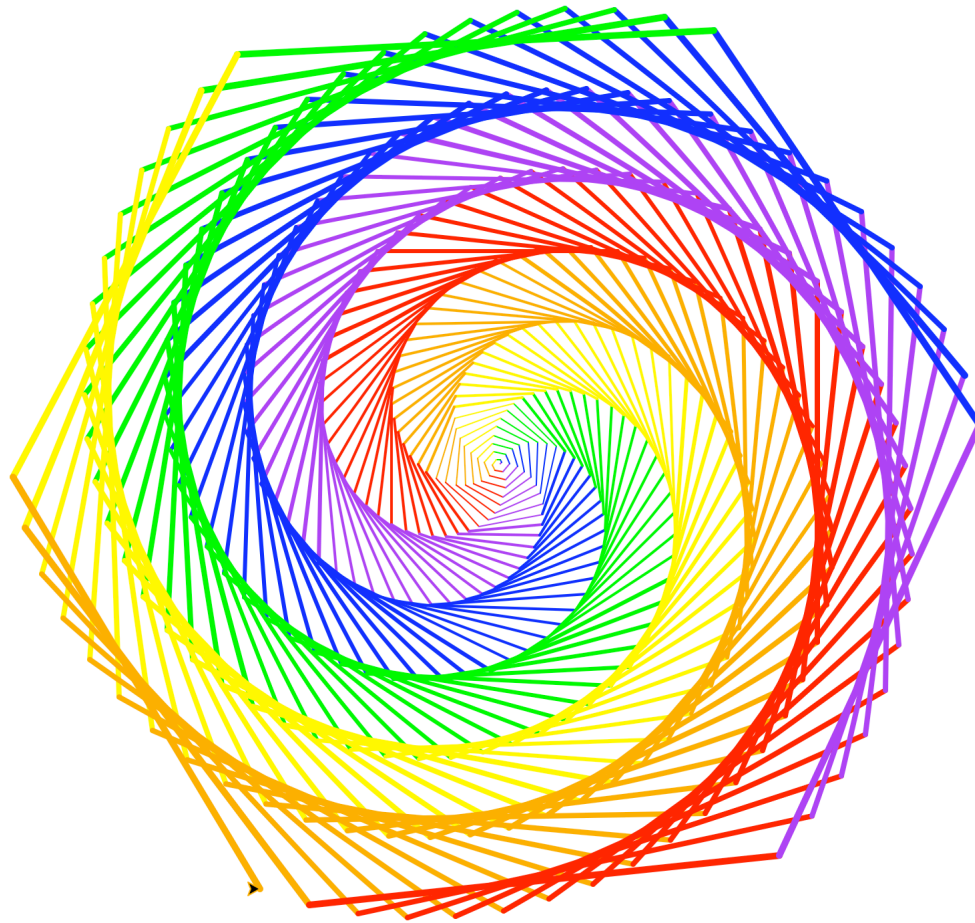


# Example: Simple.py DrawPicture

```
def drawPicture(turt):  
    turt.forward(50)  
    turt.left(90)  
    turt.forward(80)  
    turt.pencolor('red')  
    turt.right(60)  
    turt.forward(100)  
    turt.pencolor('green')  
    turt.left(60)  
    turt.forward(50)  
    turt.left(90)  
    turt.forward(200)
```



# Run Turtle, Run



# What are key concepts in Spiro.py?

```
8  import turtle
9
10 def draw(turt):
11     colors = ['red', 'purple', 'blue', 'green', 'yellow', 'orange']
12     turt.speed(0)
13     for x in range(360):
14         turt.pencolor(colors[x % 6])
15         turt.width(x/100 + 1)
16         turt.forward(x)
17         turt.left(59)
18
19 if __name__ == '__main__':
20     win = turtle.Screen()
21     t = turtle.Turtle()
22     draw(t)
23     win.exitonclick()
```

# What are key concepts in Spiro.py?

```
8  import turtle
9
10 def draw(turt):
11     colors = ['red', 'purple', 'blue', 'green', 'yellow', 'orange']
12     turt.speed(0)
13     for x in range(360):
14         turt.pencolor(colors[x % 6])
15         turt.width(x/100 + 1)
16         turt.forward(x)
17         turt.left(59)
18
19 if __name__ == '__main__':
20     win = turtle.Screen()
21     t = turtle.Turtle()
22     draw(t)
23     win.exitonclick()
```

Import turtle

Create screen/window

Create turtle

pass turtle to function

Close on click

# What are key concepts in Spiro.py?

```
8 import turtle
9
10 def draw(turt):
11     colors = ['red', 'purple', 'blue', 'green', 'yellow', 'orange']
12     turt.speed(0)
13     for x in range(360):
14         turt.pencolor(colors[x % 6])
15         turt.width(x/100 + 1)
16         turt.forward(x)
17         turt.left(59)
18
19 if __name__ == '__main__':
20     win = turtle.Screen()
21     t = turtle.Turtle()
22     draw(t)
23     win.exitonclick()
```

1 – slowest  
10 – fastest  
0 – No animation

Loop 360 times

Change pen color, how?

Width getting bigger

Draw line

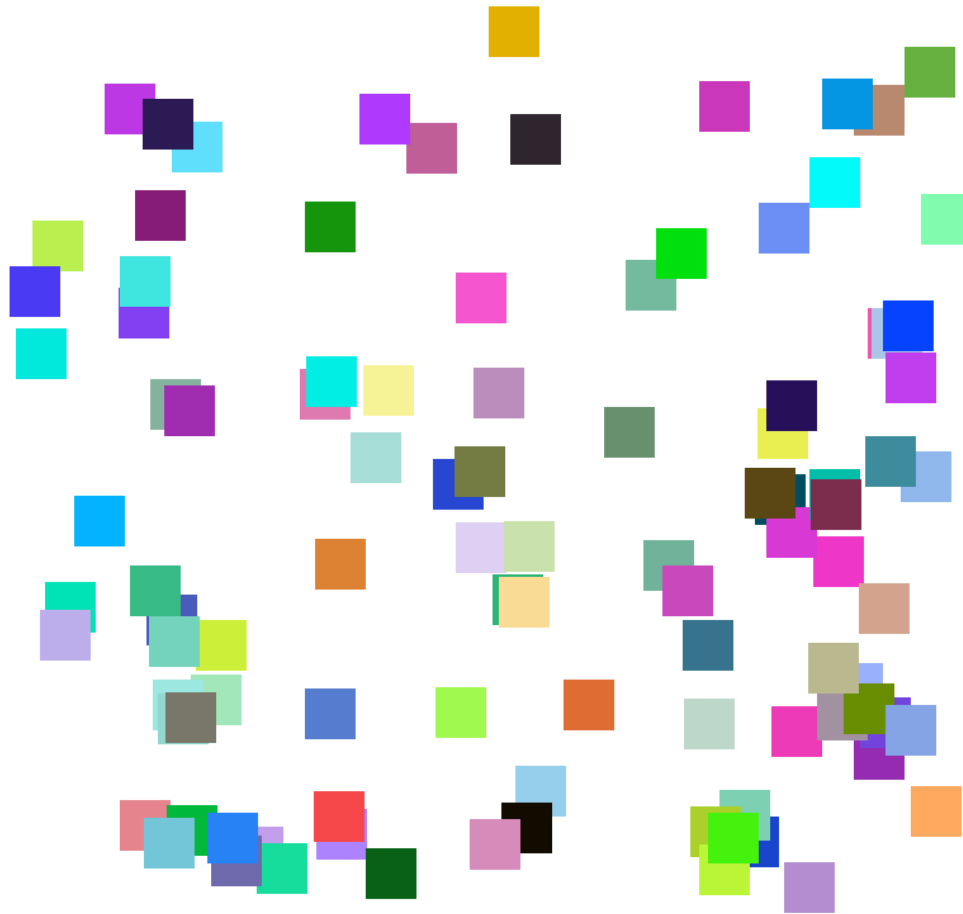
Turn

# Useful turtle functions

- **forward(n)/backward(n)** – move turtle n pixels
- **left(n)/right(n)** – turn turtle n degrees
- **pendown()/pendup()** – whether actually drawing
- **setposition(x, y)** – puts turtle in this (x,y) coordinate (a.k.a. goto, setpos)
- **sethead(n)** – points turtle in this direction (n=0 is east)
- **Many more in documentation!**
  - <https://docs.python.org/3/library/turtle.html>



# ColorMyWorld.py

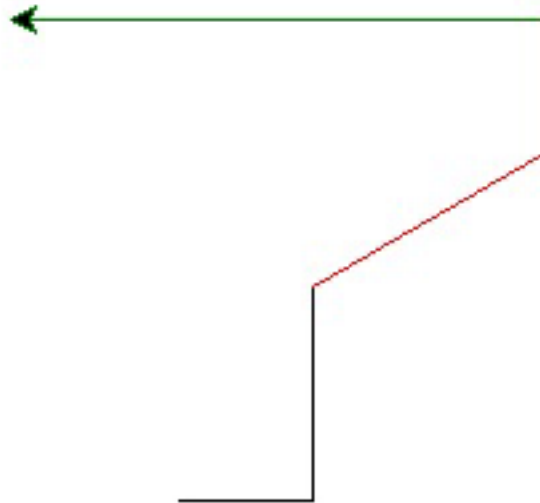


# Turtle Concepts

- **Create a screen so you can ..**
  - Exit On Click
  - Some other Screen Functions
- **Create a turtle so you can ...**
  - Move and draw using the turtle
- **Drawing Concepts**
  - Pen [up and down]
  - Fill
  - Color
  - Position

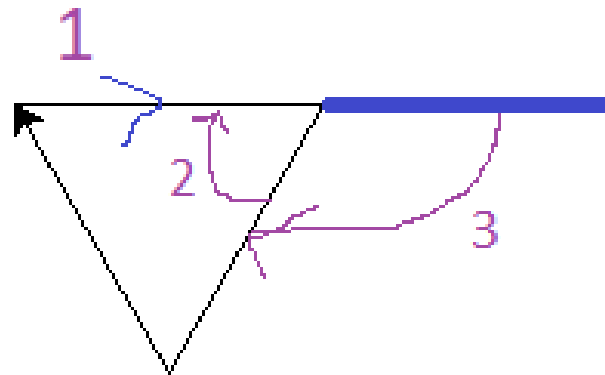
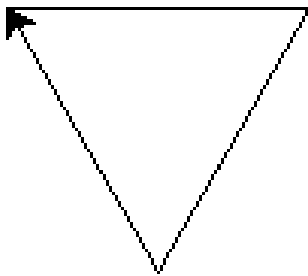
# WOTO-2 - Turtles

<http://bit.ly/10123s-0209-2>



## WOTO-2: Explanation of degrees to turn turtle to draw a triangle

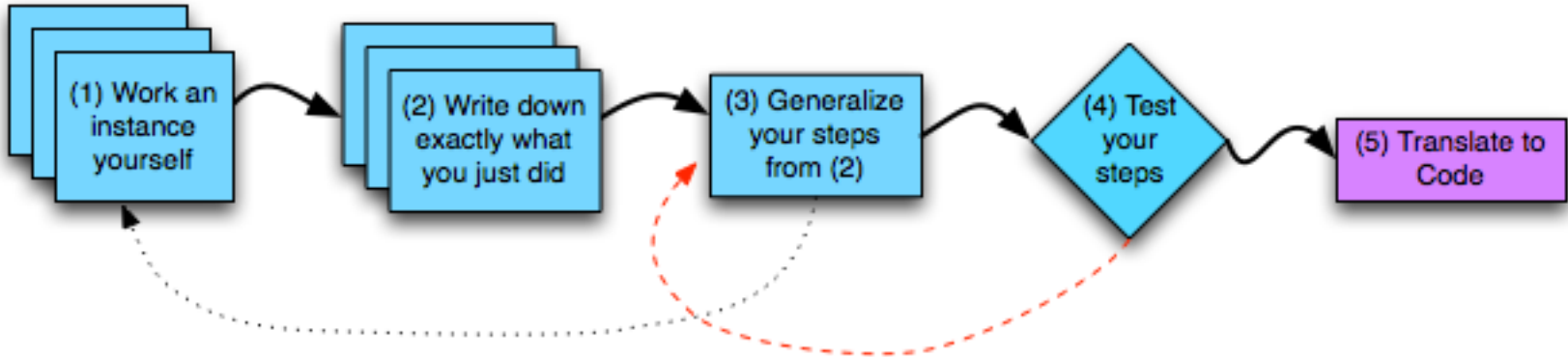
- **1) direction turtle is moving in first line**
- **2) is the angle inside the triangle which must be 60 degrees**
- **3) is the angle the turtle must turn right, which is 120 degrees**
- **4) note  $60 + 120$  is 180, which is half of a circle.**



# Looping over Sequences

- **Let's explore this:**
  - Given a sentence:
    - “Duke Computer Science is so much fun!”
  - How do we create this sentence?
    - “Dk Cmptr Scnc s s mch fn!”
  - Input is sentence. Output has vowels removed

# Designing Solution



1. **Work an instance: Go Duke -> G Dk**
2. **What did we do?**
  - a. Paper and pencil, write it down!
3. **Generalize**
4. **Test: “Computer” -> “Cmptr”?**

# What did we do?

- **“Went through character sequence and removed the vowels.”**
  - Not precise enough, what does “removed the vowels” mean? In Python?

# What did we do?

- **“Went through character sequence and removed the vowels.”**
  - Not precise enough, what does “removed the vowels” mean? In Python?
- **“For each character, if it’s not a vowel add it to the output string”**
  - “For each” -> loop
  - “if it’s not” -> if statement
  - “a vowel” -> new function?



# What did we do?

- **“Went through character sequence and removed the vowels.”**
  - Not precise enough, what does “removed the vowels” mean? In Python?
- **“For each character, if it’s not a vowel add it to the output string”**
  - “For each” -> loop
  - “if it’s not” -> if statement
  - “a vowel” -> new function?
  - “add it...” -> string concatenation

Yes because functions should only do one thing. Makes testing easier

# Step 3 and 4: Algorithm

input is string **phrase**

create a new empty string **ret**

for each character **ch** in the **phrase**

if **ch** is not a vowel

add **ch** to the end of **ret**

**ret** is the result

Step 4: Test this algorithm on “Computer”

Step 5: Now ready to translate this to code!

# Step 3 and 4: Algorithm

create a new empty string **ret**

for each character **ch** in the **phrase**

if **ch** is not a vowel



How?

add **ch** to the end of **ret**

**ret** is the result

Step 4: Test this algorithm on “Computer”

Step 5: Now ready to translate this to code!

# Step 5: Translate to Code

- **Which function do we implement first?**
  - Translate sentence?
  - Is a vowel?
  - Reasons to prefer one to the other?
- **How do we verify that our function is correct?**
  - Reproducible testing not detailed here
  - Testing is really, really important

# WOTO-3

<http://bit.ly/10123s-0209-3>

- **Is it a vowel?**

# IsVowel Functions

16 2 ^

```
def isVowel1(ch):  
    return ch in 'aeiouAEIOU'  
  
def isVowel2(ch):  
    return 'aeiouAEIOU'.count(ch) > 0  
  
def isVowel3(ch):  
    if ch == 'a' or ch == 'e' or ch == 'i' or ch == 'o' or ch == 'u':  
        return True  
    if ch == 'A' or ch == 'E' or ch == 'I' or ch == 'O' or ch == 'U':  
        return True  
    return False
```

# Testing IsVowelFunctions

```
if __name__ == '__main__':  
    print("Testing isVowel functions")  
    lstIsVowel = [isVowel1, isVowel2, isVowel3]  
  
    for v in lstIsVowel:  
        print("\nTesting: ", v)  
        print(v('a') == True)  
        print(v('E') == True)  
        print(v('b') == False)  
        print(v('Z') == False)
```

# Testing IsVowelFunctions

```
if __name__ == '__main__':  
    print("Testing isVowel functions")  
    lstIsVowel = [isVowel1, isVowel2, isVowel3]  
    for v in lstIsVowel:  
        print("\nTesting: ", v)  
        print(v('a') == True)  
        print(v('E') == True)  
        print(v('b') == False)  
        print(v('Z') == False)
```

What type is v?

v is a  
function

Call  
function v

v is isVowel1  
function the first  
time through the  
loop, then v is  
isVowel2, then v is  
isVowel3



# Accumulator Pattern: NoVowels

- **“For each character, if it’s not a vowel add it to the output string”**
- **Accumulator pattern: change a variable in a loop**
  - Accumulate a value while iterating through loop

# Accumulator Pattern: NoVowels

create a new empty string **ret**

for each character **ch** in the **phrase**

if **ch** is not a vowel

add **ch** to the end of **ret**

**ret** is the result

```
def noVowels(phrase):  
    ret = ""  
    for ch in phrase:  
        if not isVowel1(ch):  
            ret = ret + ch  
    return ret
```

# Accumulator Pattern: NoVowels

## Build new string: ret

```
def noVowels(phrase):  
    ret = ""  
    for ch in phrase:  
        if not isVowel1(ch):  
            ret = ret + ch  
    return ret
```

Initialize before loop

Update inside loop

Do something with  
value after loop