Program Design and Debugging

How do programmers *start*?

- How do you get started with a program?
- “Programming is all about debugging a blank piece of paper.” – Gerald Sussman

Top-down method

- Figure out what has to be done.
  - These are called the *requirements*
- Refine the *requirements* until they describe, in English, what needs to be done in the program.
  - Keep refining until you know how to write the program code for each statement in English.
- Step-by-step, convert the English requirements into program code.

Top-down Example

- Write a function called **pay** that takes in as input a number of hours worked and the hourly rate to be paid. Compute the gross pay as the hours times the rate. But then compute a taxable amount.
- If the pay is< 100, charge a tax of 0.25
- If the pay is >= 100 and < 300, tax rate is 0.35
- If the pay is >=300 and < 400, tax rate is 0.45
- If the pay is >= 400, tax rate is 0.50
- Print the gross pay and the net pay (gross – taxable amount).

Top-down Example: Refine into steps you can code

- Write a function called **pay** that takes in as input a number of hours worked and the hourly rate to be paid.
- Compute the gross pay as the hours times the rate.
- If the pay is< 100, charge a tax of 0.25
- If the pay is >= 100 and < 300, tax rate is 0.35
- If the pay is >=300 and < 400, tax rate is 0.45
- If the pay is >= 400, tax rate is 0.50
- Compute a taxable amount as tax rate * gross
- Print the gross pay and the net pay (gross – taxable amount).

Convert to program code

- √ Write a function called **pay** that takes in as input a number of hours worked and the hourly rate to be paid.
- Compute the gross pay as the hours times the rate.
- If the pay is< 100, charge a tax of 0.25
- If the pay is >= 100 and < 300, tax rate is 0.35
- If the pay is >=300 and < 400, tax rate is 0.45
- If the pay is >= 400, tax rate is 0.50
- Compute a taxable amount as tax rate * gross
- Print the gross pay and the net pay (gross – taxable amount).

```
def pay(hours,rate):
```

Convert to program code

- √ Write a function called **pay** that takes in as input a number of hours worked and the hourly rate to be paid.
- √ Compute the gross pay as the hours times the rate.
- If the pay is< 100, charge a tax of 0.25
- If the pay is >= 100 and < 300, tax rate is 0.35
- If the pay is >=300 and < 400, tax rate is 0.45
- If the pay is >= 400, tax rate is 0.50
- Compute a taxable amount as tax rate * gross
- Print the gross pay and the net pay (gross – taxable amount).

```
def pay(hours,rate):
  gross = hours * rate
```

Convert to program code

- √ Write a function called **pay** that takes in as input a number of hours worked and the hourly rate to be paid.
- √ Compute the gross pay as the hours times the rate.
- √ If the pay is< 100, charge a tax of 0.25
- If the pay is >= 100 and < 300, tax rate is 0.35
- If the pay is >=300 and < 400, tax rate is 0.45
- If the pay is >= 400, tax rate is 0.50
- Compute a taxable amount as tax rate * gross
- Print the gross pay and the net pay (gross – taxable amount).

```
def pay(hours,rate):
  gross = hours * rate
  if pay < 100:
    tax = 0.25
```

Convert to program code

- √ Write a function called **pay** that takes in as input a number of hours worked and the hourly rate to be paid.
- √ Compute the gross pay as the hours times the rate.
- √ If the pay is< 100, charge a tax of 0.25
- √ If the pay is >= 100 and < 300, tax rate is 0.35
- √ If the pay is >=300 and < 400, tax rate is 0.45
- √ If the pay is >= 400, tax rate is 0.50
- Compute a taxable amount as tax rate * gross
- Print the gross pay and the net pay (gross – taxable amount).

```
def pay(hours,rate):
  gross = hours * rate
  if pay < 100:
    tax = 0.25
  if 100 <= pay < 300:
    tax = 0.35
  if 300 <= pay < 400:
    tax = 0.45
  if pay >= 400:
    tax = 0.50
```

Convert to program code

- √ Write a function called **pay** that takes in as input a number of hours worked and the hourly rate to be paid.
- √ Compute the gross pay as the hours times the rate.
- √ If the pay is< 100, charge a tax of 0.25
- √ If the pay is >= 100 and < 300, tax rate is 0.35
- √ If the pay is >=300 and < 400, tax rate is 0.45
- √ If the pay is >= 400, tax rate is 0.50
- √ Compute a taxable amount as tax rate * gross
- Print the gross pay and the net pay (gross – taxable amount).

```
def pay(hours,rate):
  gross = hours * rate
  if pay < 100:
    tax = 0.25
  if 100 <= pay < 300:
    tax = 0.35
  if 300 <= pay < 400:
    tax = 0.45
  if pay >= 400:
    tax = 0.50
  taxableAmount = gross * tax
```

Convert to program code

- √ Write a function called **pay** that takes in as input a number of hours worked and the hourly rate to be paid.
- √ Compute the gross pay as the hours times the rate.
- √ If the pay is< 100, charge a tax of 0.25
- √ If the pay is >= 100 and < 300, tax rate is 0.35
- √ If the pay is >=300 and < 400, tax rate is 0.45
- √ If the pay is >= 400, tax rate is 0.50
- √ Compute a taxable amount as tax rate * gross
- √ Print the gross pay and the net pay (gross – taxable amount).

```
def pay(hours,rate):
  gross = hours * rate
  if pay < 100:
    tax = 0.25
  if 100 <= pay < 300:
    tax = 0.35
  if 300 <= pay < 400:
    tax = 0.45
  if pay >= 400:
    tax = 0.50
  taxableAmount = gross * tax
  print "Gross pay:",gross
  print "Net pay:",gross-
     taxableAmount
```

Why “top-down”?

- We start from the highest level of abstraction
  - The requirements
- And work our way down to the most specificity
  - To the code
- The opposite is “bottom-up”
- Top-down is the most common way that professionals design.
  - It provides a well-defined process and can be tested throughout.

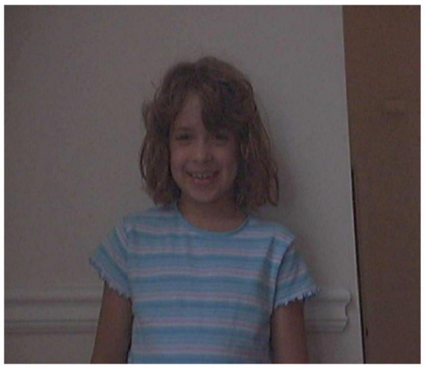
What’s “bottom-up”?

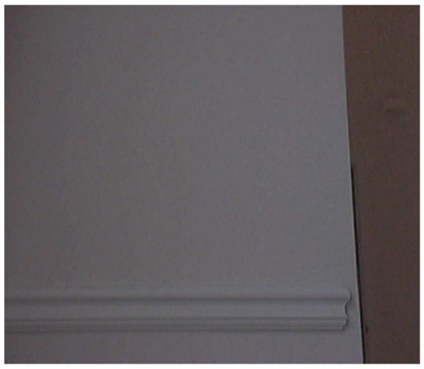
- Start with what you know, and keep adding to it until you’ve got your program.
- You *frequently* refer to programs you know.
  - Frankly, you’re looking for as many pieces you can steal as possible!

Background subtraction

- Let’s say that you have a picture of someone, and a picture of the same place (same background) without the someone there, could you subtract out the background and leave the picture of the person?
- Maybe even change the background?
- Let’s take that as our problem!

Person (Katie) and Background



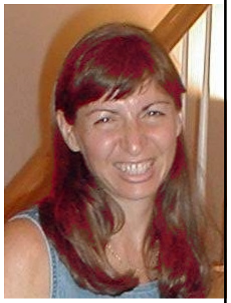


Bottom-up: Where do we start?

- What we most need to do is to figure out whether the pixel in the Person shot is the same as the in the Background shot.
- Will they be the EXACT same color? Probably not.
- So, we'll need some way of figuring out if two colors are close…

Remember this?

```
def turnRed():
 brown = makeColor(57,16,8)
 file = r“c:\mediasources\barbara.jpg"
 picture=makePicture(file)
 for px in getPixels(picture):
  color = getColor(px)
  if distance(color,brown)<50.0:
     redness=getRed(px)*1.5
    setRed(px,redness)
 show(picture)
 return(picture)
```



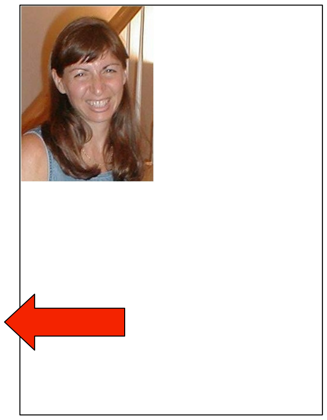
Original:

Using distance

- So we know that we want to ask:
  `if distance(personColor,bgColor) > someValue`
- And what do we then?
  - We want to grab the color from another background (a new background) at the same point.
  - Do we have any examples of doing that?

Copying Barb to a canvas

```
def copyBarb():
 # Set up the source and target pictures
 barbf=getMediaPath("barbara.jpg")
 barb = makePicture(barbf)
 canvasf = getMediaPath("7inX95in.jpg")
 canvas = makePicture(canvasf)
 # Now, do the actual copying
 targetX = 1
 for sourceX in range(1,getWidth(barb)):
   targetY = 1
   for sourceY in range(1,getHeight(barb)):
     color = getColor(getPixel(barb,sourceX,sourceY))
     setColor(getPixel(canvas,targetX,targetY), color)
     targetY = targetY + 1
   targetX = targetX + 1
 show(barb)
 show(canvas)
 return canvas
```



Where we are so far:

```
if distance(personColor,bgColor) > someValue:
  bgcolor = getColor(getPixel(newBg,x,y))
  setColor(getPixel(person,x,y), bgcolor)
```

- What else do we need?
  - We need to get all these variables set up
    - We need to input a person picture, a background (background without person), and a new background.
    - We need a loop where x and y are the right values
    - We have to figure out personColor and bgColor

Swap a background using background subtraction

```
def swapbg(person, bg, newbg):









      if distance(personColor,bgColor) > someValue:
        bgcolor = getColor(getPixel(newbg,x,y))
        setColor(getPixel(person,x,y), bgcolor)
```

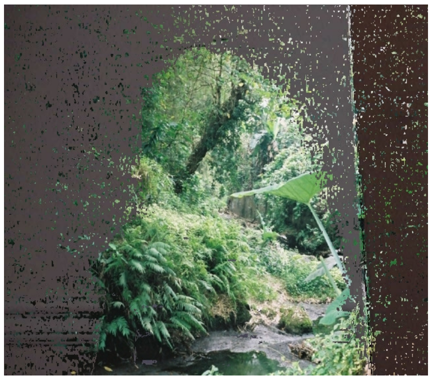
Swap a background using background subtraction

```
def swapbg(person, bg, newbg):
  for x in range(1,getWidth(person)):
    for y in range(1,getHeight(person)):
      personPixel = getPixel(person,x,y)
      bgpx = getPixel(bg,x,y)
      personColor= getColor(personPixel)
      bgColor = getColor(bgpx)
      if distance(personColor,bgColor) > someValue:
        bgcolor = getColor(getPixel(newbg,x,y))
        setColor(getPixel(person,x,y), bgcolor)
```

Simplifying a little, and specifying a little

```
def swapbg(person, bg, newbg):
  for x in range(1,getWidth(person)):
    for y in range(1,getHeight(person)):
      personPixel = getPixel(person,x,y)
      bgpx = getPixel(bg,x,y)
      personColor= getColor(personPixel)
      bgColor = getColor(bgpx)
      if distance(personColor,bgColor) > 10:
        bgcolor = getColor(getPixel(newbg,x,y))
        setColor(personPixel, bgcolor)
```

Trying it with a jungle background



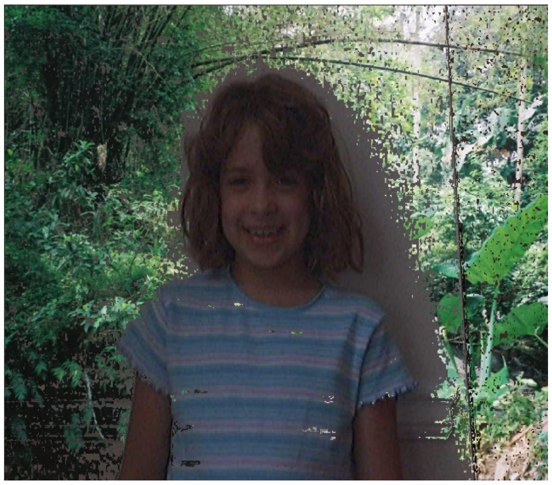
What happened?

- It looks like we reversed the swap
  - If the distance is great, we want to KEEP the pixel.
  - If the distance is small (it’s basically the same thing), we want to get the NEW pixel.

Reversing the swap

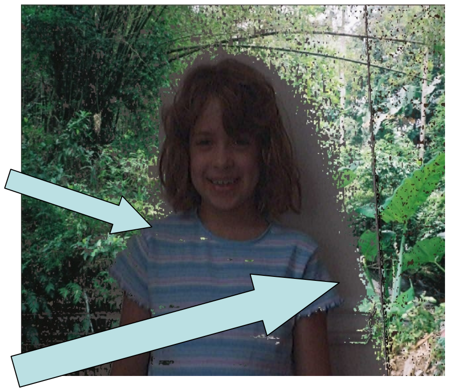
```
def swapbg(person, bg, newbg):
  for x in range(1,getWidth(person)):
    for y in range(1,getHeight(person)):
      personPixel = getPixel(person,x,y)
      bgpx = getPixel(bg,x,y)
      personColor= getColor(personPixel)
      bgColor = getColor(bgpx)
      if distance(personColor,bgColor) < 10:
        bgcolor = getColor(getPixel(newbg,x,y))
        setColor(personPixel, bgcolor)
```

Better!



But why isn't it a lot better?

- We've got places where we got pixels swapped that we didn't want to swap
  - See Katie's shirt stripes
- We've got places where we want pixels swapped, but didn't get them swapped
  - See where Katie made a shadow



How could we make it better?

- What could we change in the program?
  - We could change the threshold “someValue”
  - If we increase it, we get *fewer* pixels matching
    - That won’t help with the shadow
  - If we decrease it, we get *more* pixels matching
    - That won’t help with the stripe
- What could we change in the pictures?
  - Take them in better light, less shadow
  - Make sure that the person isn’t wearing clothes near the background colors.

Side trip: This is Debugging, too!

- Debugging is figuring out what your program is doing, what you want it to do, and how to get it from where you are to where you want it to be.
- When you get error messages, that’s the *easy* kind of debugging!
  - You know that you just have to figure out what Python is complaining about, and change it so that Python doesn’t complain anymore!
- The harder kind is when the program *works*, but you *still* don’t know why it’s not doing what you want.
- **First step in any debugging:** *Figure out what the program is doing!*
  - This is true if you have errors or not.
  - If you have errors, the issues are:
    - Why did it work *up to there*?
    - What are the values of the variables at that point?

How to understand a program

- Step 1: *Walk* the program
  - Figure out what every line is doing, and what every variable's value is.
  - At least, do this for the lines that are confusing to you.
- Step 2: *Run* the program
  - Does it do what you think it's doing?
- Step 3: *Check* the program
  - Insert ***print*** statements to figure out what values are what in the program
  - You can also use print statements to print out values to figure out how IF's are working.

How to understand a program

- Use the command area!
  - Type commands to check on values, to see how functions work.
  - Use showVars() to help, too.
- Step 4: *Change* the program
  - Now, change the program in some interesting way
    - Instead of all pixels, do only the pixels in part of the picture
  - Run the program again. Can you see the effect of your change?
  - If you can change the program and understand *why* your change did what it did, you understand the program

Running/Debugging Example

- Function to convert temperature from Fahrenheit to Celsius using the formula:

$$C = \frac{5(F - 32)}{9}$$

```
def fahrenheitToCelsius(fahrenheit):
    celsius = (5 / 9) * (fahrenheit - 32)
    return celsius
```

Debugging Example

```
def fahrenheitToCelsius(fahrenheit):
    celsius = (5 / 9) * (fahrenheit - 32)
    return celsius
```

1. Mentally walk through the program
2. Try it!

```
>> print fahrenheitToCelsius(100)
0
```

3. Use print statements to try to track down the error

Debugging Example

```
def fahrenheitToCelsius(fahrenheit):
    #Comment out original line of code
    #celsius = (5 / 9) * (fahrenheit - 32)
    conversionFactor = 5 / 9
    tempF = (fahrenheit - 32)

    print "Fahrenheit - 32 = " , tempF
    print "Conversion = " , conversionFactor
    celsius = conversionFactor * tempF
    print "Celsius = ", celsius
    return celsius
```

Debugging Run

```
>>> print fahrenheitToCelsius(100)
Fahrenheit - 32 =  68
Conversion =  0
Celsius =  0
0
```

By examining the result of the print statements we have now identified the precise location of the bug.

The conversion factor is not computed correctly. Since we are setting the conversion factor to 5 / 9 this instructs the compiler to compute the division of two integers, which results in zero.

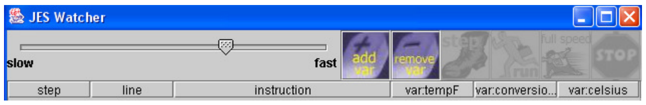
Fix: Force floating point division by using 5.0 / 9

Using the Watcher

- It can be a hassle to add lots of print statements and then remove them
- An alternate technique is to use the Watcher
  - The watcher lets you see which lines are running and when.
  - You can add variables to see their values.
  - You can change the speed of the program.
    - Faster program execution means fewer updates in the Watcher.

Using the Watcher

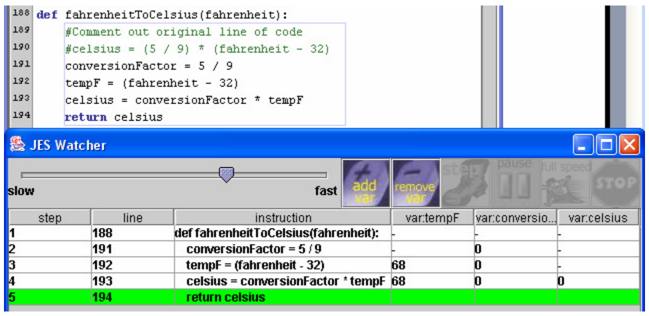
- Click on the “Watcher” button
- Add the names of any variables you want to watch



- Run your program and the values of the variables will show in the Watcher window line by line

Modified Program with Bug

```
def fahrenheitToCelsius(fahrenheit):
    #Comment out original line of code
    #celsius = (5 / 9) * (fahrenheit - 32)
    conversionFactor = 5 / 9
    tempF = (fahrenheit - 32)
    celsius = conversionFactor * tempF
    return celsius
```



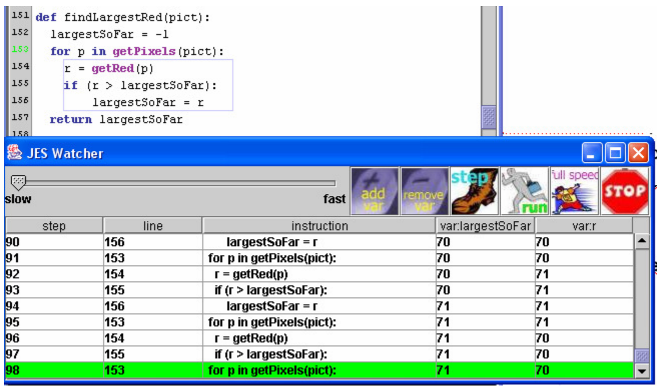
```
def fahrenheitToCelsius(fahrenheit):
    #Comment out original line of code
    #celsius = (5 / 9) * (fahrenheit - 32)
    conversionFactor = 5 / 9
    tempF = (fahrenheit - 32)
    celsius = conversionFactor * tempF
    return celsius
```

JES Watcher

step	line	instruction	var:tempF	var:conversio...	var:celsius
1	188	def fahrenheitToCelsius(fahrenheit):	-	-	-
2	191	conversionFactor = 5 / 9	-	0	-
3	192	tempF = (fahrenheit - 32)	68	0	-
4	193	celsius = conversionFactor * tempF	68	0	0
5	194	return celsius			

Stepping, Stopping

- Useful in loops to watch variables



```
def findLargestRed(pict):
  largestSoFar = -1
  for p in getPixels(pict):
    r = getRed(p)
    if (r > largestSoFar):
        largestSoFar = r
  return largestSoFar
```

JES Watcher

step	line	instruction	var:largestSoFar	var:r
90	156	largestSoFar = r	70	70
91	153	for p in getPixels(pict):	70	70
92	154	r = getRed(p)	70	71
93	155	if (r > largestSoFar):	70	71
94	156	largestSoFar = r	71	71
95	153	for p in getPixels(pict):	71	71
96	154	r = getRed(p)	71	70
97	155	if (r > largestSoFar):	71	70
98	153	for p in getPixels(pict):	71	70

Debugging

- It takes some time getting used to the Watcher, but the time spent will pay off in time saved later
- At a minimum, get used to debugging programs using print statements

Designing and Debugging

- Most important hint on designing: Start from previous programs!
  - The best designers don't start from scratch.
- Most important hint on debugging: *Understand* your program.
  - Know what each line is doing.
  - Know what *you meant* for each line to be doing.
  - Try *lots* of examples.