



# COMP2720: Automating Tools for New Media

How to design and debug a  
program



# 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  $\geq 100$  and  $< 300$ , tax rate is 0.35
- If the pay is  $\geq 300$  and  $< 400$ , tax rate is 0.45
- If the pay is  $\geq 400$ , 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  $\geq 100$  and  $< 300$ , tax rate is 0.35
- If the pay is  $\geq 300$  and  $< 400$ , tax rate is 0.45
- If the pay is  $\geq 400$ , tax rate is 0.50
- Compute a taxable amount as  $\text{tax rate} * \text{gross}$
- Print the gross pay and the net pay (gross – taxable amount).

# Convert to program code – Step 1

- √ 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  $\geq 100$  and  $< 300$ , tax rate is 0.35
- If the pay is  $\geq 300$  and  $< 400$ , tax rate is 0.45
- If the pay is  $\geq 400$ , tax rate is 0.50
- Compute a taxable amount as  $\text{tax rate} * \text{gross}$
- Print the gross pay and the net pay (gross – taxable amount).

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

# Convert to program code – Step 2

√ 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  $\geq 100$  and  $< 300$ , tax rate is 0.35

If the pay is  $\geq 300$  and  $< 400$ , tax rate is 0.45

If the pay is  $\geq 400$ , tax rate is 0.50

Compute a taxable amount as  $\text{tax rate} * \text{gross}$

Print the gross pay and the net pay (gross – taxable amount).

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

# Convert to program code – Step 3

√ 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  $\geq 100$  and  $< 300$ , tax rate is 0.35

If the pay is  $\geq 300$  and  $< 400$ , tax rate is 0.45

If the pay is  $\geq 400$ , tax rate is 0.50

Compute a taxable amount as  $\text{tax rate} * \text{gross}$

Print the gross pay and the net pay (gross – taxable amount).

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

# Convert to program code – Step 4

- √ 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  $\geq 100$  and  $< 300$ , tax rate is 0.35
  - √ If the pay is  $\geq 300$  and  $< 400$ , tax rate is 0.45
  - √ If the pay is  $\geq 400$ , tax rate is 0.50
- Compute a taxable amount as  $\text{tax rate} * \text{gross}$
- Print the gross pay and the net pay (gross – taxable amount).

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

# Convert to program code – Step 5

- √ 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  $\geq 100$  and  $< 300$ , tax rate is 0.35
  - √ If the pay is  $\geq 300$  and  $< 400$ , tax rate is 0.45
  - √ If the pay is  $\geq 400$ , tax rate is 0.50
  - √ Compute a taxable amount as  $\text{tax rate} * \text{gross}$
- Print the gross pay and the net pay (gross – taxable amount).

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

# Convert to program code – Step 6

- √ 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  $\geq 100$  and  $< 300$ , tax rate is 0.35
- √ If the pay is  $\geq 300$  and  $< 400$ , tax rate is 0.45
- √ If the pay is  $\geq 400$ , tax rate is 0.50
- √ Compute a taxable amount as  $\text{tax rate} * \text{gross}$
- √ Print the gross pay and the net pay ( $\text{gross} - \text{taxable amount}$ ).

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

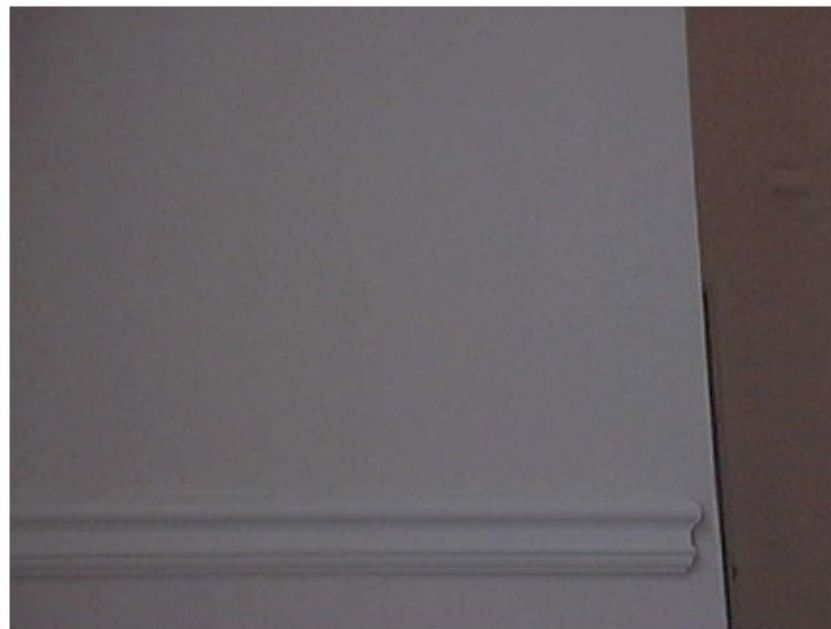
- Start with what you know, and keep adding to it until you've got your program.
- You frequently refer to programs you know.
  - Frankly, you're looking for as many pieces you can steal as possible!



# Example: 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



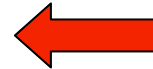


## Bottom-up: Where do we start?

- 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"E:\comp2720\images\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)
```



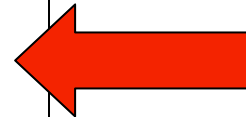
Original:

# 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 colour 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)  
    canvas = makeEmptyPicture(400,600)  
    # 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):
    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

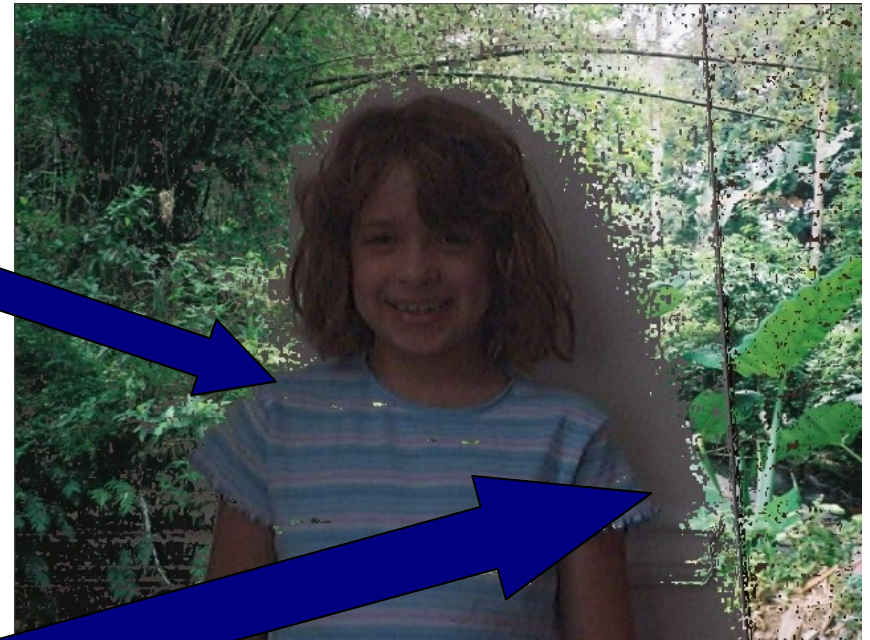
```
def swapbg(person, bg, newbg):  
    for x in range(1, getWidth(person)):  
        for y in range(1, getHeight(person)):  
            personPixel = getPixel(person, x, y)  
            bgpx = getPixel(bg, x, y)  
            personColor = getColor(personPixel)  
            bgColor = getColor(bgpx)  
            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 decrease it, we get *fewer* pixels matching.
    - That won’t help with the shadow.
  - If we increase 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 - 1

- 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 (use `printNow` to print while program is running).
  - You can also use print statements to print out values like `getRed` to figure out how if's are working.

# How to understand a program - 2

- Use the command area!
  - Type commands to check on values, to see how functions work.
  - Not sure what `getPixel` does? Try it!
  - 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.

# Use the JES 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**.



The screenshot shows the JES Debugger window. At the top, there is a title bar 'JES Debugger' and a toolbar with buttons for 'slow', 'fast', 'add var', 'remove var', 'pause', and 'STOP'. Below the toolbar is a table with the following columns: 'step', 'line', 'instruction', 'var:item', and 'var:sum'. The table contains 12 rows of data, with the last row (step 12) highlighted in green.

step	line	instruction	var:item	var:sum
0	1	def sumlist(list):	-	-
1	2	sum = 0	-	0
2	3	for item in list:	-	0
3	3	for item in list:	4	0
4	4	sum = sum + item	4	4
5	3	for item in list:	5	4
6	4	sum = sum + item	5	9
7	3	for item in list:	2	9
8	4	sum = sum + item	2	11
9	3	for item in list:	1	11
10	4	sum = sum + item	1	12
11	3	for item in list:	1	12
12	5	return sum	-	-

# Another example: Chromakey

- Have a background of a *known* colour.
  - Some color that won't be on the person you want to mask out.
  - Pure green or pure blue is most often used.
  - Here a blue bed sheet is used.
- This is how the weather people seem to be in front of a map—they're actually in front of a blue sheet.



# Chromakey recipe

```
def chromakey(source,bg):  
    # source should have something in front of blue, bg is the new background  
    for x in range(1,getWidth(source)):  
        for y in range(1,getHeight(source)):  
            p = getPixel(source,x,y)  
            # My definition of blue: If the redness + greenness < blueness  
            if (getRed(p) + getGreen(p) < getBlue(p)):  
                #Then, grab the color at the same spot from the new background  
                setColor(p,getColor(getPixel(bg,x,y)))
```

# Can also do this with getPixels()

```
def chromakey2(source,bg):  
    # source should have something in front of blue, bg is the new background  
    for p in getPixels(source):  
        # My definition of blue: If the redness + greenness < blueness  
        if (getRed(p) + getGreen(p) < getBlue(p)):  
            #Then, grab the color at the same spot from the new background  
            setColor(p,getColor(getPixel(bg,getX(p),getY(p))))
```

# Example results





# The value of “sub-recipes”: Algorithms

- Algorithms are ways of doing things.
  - Independent of the programming language used.
  - Independent also of the data being used.
- We’ve seen some sub-recipes/algorithms:
  - Copying pixels/samples.
  - Sampling pixels/samples.
- Algorithms are useful tools in your bag of tricks.
  - They offer worked out ways-of-doing-things.



# 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.
  - Comment your programs.



# Tips on background subtraction (and chromakey)

- Use a tripod!
  - If you're off by a pixel, you're way off
- For background subtraction, use a boring background
  - Complex backgrounds have a lot more pixel variation, so it's harder to make sure that you're close
- For chromakey, use a background color that's
  - Primary.
  - Unusual. (Not red! You've got lots of red in your face!)
- You may have to tweak your color definition to work with the kind of color and context you have.



## What to do now..

- Read chapter 9 in text book
- Assignment 1: Try to submit your program and picture files early – and check your submissions in STReaMS.
- You can submit each file many times.