# Homework 09 - Transcript-Summary-Conclusion Table & Appendix

Andrew Carter and Beryl Egerter

April 3, 2013

## 1 Transcript-Summary-Conclusion Table

### 1.1 A note on formatting

Following are annotated transcripts of ACBE4Q1, ACBE3Q1, ACBE3Q4, and ACBE1Q3. The assignment asks for these in a table format. Due to the wordy nature of the transcripts and the conclusions (and partially the summaries), we found the table format which restricts the width of each of these categories to less than a third of the page extremely cramped and almost impossible to read. In the interest of readability we have provided these in a format of:
*[time]*
*Speaker:*
    *transcript*
**Context**: any context required
**Summary**: brief summary of corresponding transcript
**Conclusions**: conclusions drawn from corresponding transcript

### 1.2 High Level Points

The high level points we are trying to make with these transcripts are:

- (ACBE4Q1) Execution order appears to work better for the evaluation questions we have provided. Even by starting using a different method, a participant ended up executing the code instead.
- (ACBE3Q1 & ACBE3Q4)The right method can help a student move past concepts they are not familiar with. While evaluating, a student used execution order which helped him understand a concept, and switching to top down let him ignore a large portion of the code and clear up some misunderstandings.
- (ACBE1Q3) Top down order on a debugging question while not understanding many concepts used in the code can lead to someone forgetting the question.

### 1.3 A note on context

In the context of each segment of transcript, line numbers are referred to. These line numbers correspond to line numbers of code within the Appendix (Section 2 of this document). Also used are figures which are scanned copies of the paper that the students wrote on during the interviews. The figures follow the annotated transcripts as Section 1.5 of this document.

## 1.4 Annotated Transcripts

## 1.4.1 ACBE4Q1

All line numbers refer to the code of Question 1.1 found in the Appendix.

*[2:47-2:56]:*
*S:*

*So this first think I'm looking to do is to find some of the basic functions, functions that I can identify first, that don't call other functions.*

**Summary:**

The student describes his plan of attack for this problem.

**Conclusion:**

The students first choice for this problem is to go after simple questions, he does that without looking very closely at the problem given to him yet.

*[2:56-3:00]*
*S:*

*"For example func4 here just multiplies a with b."*

**Context:** Lines 3-4

**Summary:**

The student identifies func4 as a simple function, and correctly defines its semantics.

**Conclusion:**

This code does have some simple functions, and the student can simplify these functions.

[3:00-3:22]
S:

So now I'm looking to see if there are any other things
Function 1 looks like it calls other functions for some "f"s.
Oh it takes some "f" here, which presumably is a function, and then calls that.
I'll come to that later, just because I'm not sure.

**Context:** Lines 6,9 (and implicitly 6-10)

**Summary:**

The student identifies another function, but passing a function by a pointer trips him up initially. He comes up with a hypothesis, that the function is being passed in, but is unsure of himself. Furthermore, he is unable to generalize that function, even though it does not reference any other functions internally, due to the function being passed.

Conclusion:

This function, as we'll see later, is identifiable as a map followed by a sum, without a need to reference other functions. However at this time, the student is unwilling to follow through with his initial strategy on this particular function.

*[3:22-3:37]*

*S:*

*Main is going to call function3, function3 is going to call function2, function2 calls function1 of the list, and function1 is the messy function.*

**Context:** Line 12-13, 15-16, 0-1

**Summary:**

The student switches to following the execution of the program starting at main. He ends up at function1, which describes as messy.

**Conclusion:**

The student appears to have been going from the top down, and switches to execution order the instant he sees a function that should be called first. It appears that he looked at main without looking through func3, but it is possible that he identified func3 as being complex and decided to ignore it, however that hypothesis is contradicted by his next statement.

*[3:41-3:52]*

*S:*

*Ok, so I want to replace, um..., [3:47] I'm gonna just, everywhere there is a function4, I'm just going to replace that with a multiply operator.*

**Context:** Line 1, Figure X

**Summary:**

He crosses out func4 on line 1 and replaces it with '*'.

**Conclusions:**

He switches back to gathering the results of simplifying some of the functions, the only function he successfully simplified was func4. This implies he was not done with going through all of the functions when he hit main, and in fact reaching main keyed him into following execution order, before returning to his search for simple functions.

*[3:52-4:25]*

*S:*

*And so now I just going to try to trace the code path.*
*So I'm going to have function 3 maybe, kinda give,*
*maybe a backtrace of the arguments being passed in.*
*Right now we have func3 of list, 1, 2, 3, and 4,*

**Context:** Lines 13, Figure X

**Summary:**

The student goes back to execution order, and starts writing out all of the function calls starting with the call from main function 3.

**Conclusion:**

The student is switching strategies from simple functions to execution order.

*[4:25-4:50]*
*S:*

*so now I'm going to look into 1,2,3,4*
*now that I see that func3 is just passing in this list to function 2.*
*So func2 is going to have the list 1,2,3,4 and its also going to have a 4,*
*I'm assuming that as actually going to be the function we call,*
*because I remember seeing a func4.*

**Context:** Lines 15-16, Figure X

**Summary:**

The student picks up on the fact that func3 just adds another argument. He appears to be confused about the difference between func4 and the number 4. However he is hesitant in his statement.

**Conclusion:**

A possible explanation is that his analysis and subsequent crossing out of func4 as an argument and trying to figure out how "f" works in func1 has primed him for trying to pass a function as an argument, even if it just happens to be a number. This could be dangerous for some students, and may be a risk for uncovering unfamiliar constructs before they are ready to handle them, in this case however we'll see that the student is able to correct himself without too much trouble.

*[4:50-5:02]*
*S:*

*So maybe that will be called, maybe like a map, or at this point maybe more like a fold, because we have an accumulator.*

**Context:** Line 6-10

**Summary:**

The student ponders a bit on how func1 will work. He initially identified it as a map, however changed his mind and decided it was a fold.

**Conclusion:**

Clearly the purpose of func1 has been weighing on his mind. It is unclear why he chose to go back now, perhaps he (correctly, but for the wrong reasons) decided that func4 was the same as "f" and so he had new information. Alternatively he worked some of it out in the back of his mind. Either way, interestingly enough he identifies both parts of func1 (as it is a map followed by a fold), but only separately. In short, the student does decide to fully decipher the purpose of func1 at this time either, furthermore without looking at func2 the only information he may have gained (f=func4) was derived from the wrong source. He still has not generalized func1.

*[5:02-5:15]*
*S:*

*But, so now we are at func2 and we are going to return func1 of a list, the number, oh and func4, ok, so, the 4 wasn't actually going to be func4, but there is going to be a func4.*

**Context:** Lines 0-1

**Summary:**

The student correctly identifies the passing of func4 and amends his previous statement. He appears to be more confident in his deductions this time, as this appears to make more sense to him.

**Conclusion:**

Returning to execution order seems to have allowed the student to make more progress. Fortunately any progress he may have made deciphering the purpose of func1 due to his previous misestimation that 4 translated to func4 still probably holds true. It still would probably have been better in the general case to continue with execution order rather than skip to func1 as there may not have been a func4.

*[5:15-5:50]*
*S:*

*So we have func1 of 1,2,3,4 and our number is 4 and our function is the multiply, which I'm just going to write in as a multiply for simplicity. So now what were going to is start off with 0, now I'm just going to..., since I've identified func4 as kinda the basic thing, since f can be replaced with a multiply, I'm just going to start working with the variables here.*

**Context:** Lines 0-1, 6-7, Figure X

**Summary:**

The student finishes execution order, and starts the calculation, omitted is the transcript performing the calculations of the inner for loop, however a quote from the student sums it up quite nicely “and basic maths, weeee.”

Conclusion:

The student was unable to generalize func1, and still has to do the calculations out individually instead of generalizing the function call.

[7:30-7:49]
S:

So its going to return 40, I'm just going to trace it all the way,
because its all returns I'm guessing its just going to be 40 at the end.
But, umm, accumulator from func1 goes to func2,
func2 is called from func3, return, return, return, and so we're going to print it.
So I'm thinking its going to print 40.

**Context:** Lines 10, 1, 16, 13

**Summary:**

After finishing his calculations, he unwinds the call stack and arrives at the correct answer.

**Conclusions:**

The student finished the problem, the bulk of the work appeared to be done while following execution. It is unclear that delaying solving func4 would have hurt his analysis any, but all of his attempts at generalizing func1 appear to have been fruitless. He still feels the need to return up the call stack, and so isn't even confidant in generalizing the intermediary functions as just adding arguments.

*[7:50-7:57]*
*A :*

*Ok, can you give us a general high level description of what func2 does?*
*[7:57] So func2 is a function that takes a list and number,*
*[8:02] ch,ch,ch,ch,ch,chewwwww*
*[8:07] and essentially maps function4 across the list, and then folds that...*
*Sorry, uhh..*
*[8:17]*
*[8:22] so it maps function4 the list with num as kind of the second argument,*
*and then folds the list by addition, is what it seems to do for me.*
*[8:40]*

**Context:** Lines 0-1, 6-10

**Summary:**

The student takes a while to get through the explanation of what func2.

**Conclusion:**

Clearly he came up with neither a high level explanation of what func1 nor what func2 did. However when pressed he is able to come up with an explanation.

*[8:50-9:02]*
*A :*

*I have one more question,*
*if num were 1 is there a simpler python function that you could use in that case,*
*in the special case where num equals 1.*

*[9:02-9:16]*
*S:*

*I think it would be sum of the list, if sum works on a list,*
*[9:06] you could also just fold plus on the list,*
*but I think there is just a sum command for a list in python.*

**Context:** Lines 0-1, 6-10

**Summary:**

The student correctly explains what would happen if we changed the 4 being passed to a 1.

**Conclusion:**

Here the student answers the question immediately (he spends more time wondering if sum can take a list of arguments), after the previous question he understands how func2 (and to some extent func1) work well enough to understand how simple modifications would affect the output. He has created a generalization of func2.

## 1.4.2 ACBE3Q4

All line numbers refer to the code of Question 2.2 found in the Appendix.

*[18:54-19:00]:*
*S:*

> *Alright so we have a board class which is gonna presumably represent the connect 4 board... um...*

**Context**: Line 2
**Summary**: Student relates the board class to the game described in the problem statement.
**Conclusions**: Student understands the game described in the verbal prompt for this problem.

*[19:00-19:15]:*
*S:*

> *the constructor automatically sets the width to seven oh in this case it forces constraints um like arguments essentially ... and width is always going to be equal to seven, the height equal to six*

**Context**: Lines 3 -6
**Summary**: Student recognizes constructor and finds values of width and height.
**Conclusions**: Student recognizes the __init__(self) function as a constructor.

*[19:15-20:09]:*
*S:*

> *um ... and its going to create an array of arrays or a list of lists depending on what you call it in python ... um ...*
> *then it appears to not force the height to be six*
> *for i in range width ... yeah that could be cause of problems ... um*
> *i don't know if, if lists in python are dynamically allocated or not*
> *i don't think yeah i think they are, you can keep adding to them can't you...*

*I:*

> *Yes*

*S:*

> *Yeah ... so i wonder what this is doing. I guess it would be creating ... an array with at least six - er, seven secondary arrays in it*

**Context**: Line 4
**Summary**: Student is confused by arrays/lists in Python.
**Conclusions**: Student is not very familiar with lists in Python.

*[20:09-21:01]:*
*S:*

*um ... and you don't necessarily [gestures] specify the height ... i guess that doesn't matter you just need to check along the way ... um ... alright so drop*
*i guess that is where you drop a piece in ... if column less than self.width um self.board.append player*
*oh ok so its doing [starts drawing on paper] like a ... array of arrays essentially and then it just pops on a color - i don't know are they black and red? I think they are... so sorta pops on a color as they happen and that way ... oh i guess no append is going on the end isn't it ...um*

**Context**: Line 10, picture
**Summary**: Student figures out what the board data structure looks like using the drop function.
**Conclusions**: Append being used in drop function helps student realize what the lists look like and how they change. The student believes the functions have been named intelligently since he can match it up with an action in Connect Four.

*[21:01-21:04]:*
*S:*

*so this is where you would need to do the check [points at drop function with pen]*

**Context**: Lines 8 - 12
**Summary**: Student says that this function is where the bug could be fixed.
**Conclusions**: Student is confident of a solution despite only having looked through a small portion of the code.

*[21:04-21:45]:*
*S:*

*if column less than self.width then append it um*
*You could do - you could add to that if statement? let's see ... and [writing on paper] um column.size, is that a thing? oh no its going to be board at column is what it's going to be board at column*
*some sort of size operator*

*I:*

*It's len*

*S:*

*Ok, len*

**Context**: Lines 8-12, Picture
**Summary**: Student starts writing a fix for the bug during which he asks about a size operator for lists.
**Conclusions**: More evidence that student is unfamiliar with Python.

*[21:45-22:30]:*
*S:*

> *um ... less than  I have ...[mumbles] ... and board column . len less than ... height is six so it's gotta be - oh we can just do less than self.height*
> *...gotta keep good encapsulation...um ... yeah so this check would go right here [points at if in drop function]*
> *and that would keep you from ... otherwise it would return false and not allow you to drop if you exceeded the height ... um ... bounds ... that would work.*

**Context**:-Lines 8-12, Picture
**Summary**: Student finishes writing the fix.
**Conclusions**: Student has assumed that the original author of this function had good encapsulation.

### 1.4.3 ACBE1Q3

All line numbers refer to the code of Question 2.1 found in the Appendix.

*[14:04 - 15:38]:*

*A:*

*so if you want to ask us any questions about like ruby syntax or something*

*S:*

*ok*

*B:*

*yeah we can answer them*

*S:*

*what is this like um header thing, the pound..*

*A:*

*that's used on linux systems to tell it to run ruby*

*S:*

*Oh ok*

*B:*

*where ruby is*

*S:*

*Oh ok*
*What does it mean if something has a um money sign in front of it?*

*B:*

*Oh it means that it is a global variable*

*S:*

*Ok, so its a global [marks paper]*
*So there are two lists at the top*
*months thirty, months thirty one and they're global*
*and then the class our date starts*
*what's attr accessor?*

*B:*

*that means that it's got um it's got a member variable of the class and that you can access*

*S:*

*so these are the member variables?*

*B:*

*so it's like saying these are public*

*S:*

*ooooh i see*

**Context**: Lines 0-8, File 2, Picture
**Summary**: Student asks about ruby syntax such as the pound sign, the money sign, and

attr_accessor.
**Conclusions**: Student is very unfamiliar with Ruby.

*[15:38 - 16:11]:*
*S:*
*year, month, day*
*what's the at symbol?*
*B:*
*at is how I believe you specify a member variable of the class*
*A:*
*i t's like self dot in python*
*S:*
*i see*
*ok oh so this is kinda like a constructor*
*er initialize is sorta like a constructor you're just setting the variables to their values ok so [writes on paper]*

**Context**: Lines 10-14, File 2, Picture
**Summary**: Student asks about the at symbol and realizes the function is a constructor.
**Conclusions**: Given parallels between languages, the student can extrapolate the purpose of a function.

*[16:11 - 16:59]:*
*S:*
*is equal*
*takes in some variable d*
*so is this kinda like python where you don't have to specify the type?*
*A/B:*
*right/yup*
*S:*
*ok that's difficult [laughs]*
*um puts year equal to d year*
*oh i see*
*it's kinda like an if statement*
*so is equal is just checking to see if d is the same as [gestures] this?*

**Context**: Lines 16-20, File 2, Picture
**Summary**: Student compares Ruby to Python and figures out the purpose of is_equal?.
**Conclusions**: After knowing that comparing to Python worked well previously, the student

creates parallels to ask questions.

*[16:59 - 19:01]:*

*S:*

*ok [writes next to is equals], and then here's the leap year stuff*
*if year mod 400 equals 0 put 1, else if year*
*mod 100 equals 0 puts nil, is nil like null?*

*A:*

*None*

*B:*

*or False*

*S:*

*ok*

*A:*

*I think it's closer to None*

*S:*

*what does that mean? Is that the same as false?*

*A:*

*Ruby also has a false in the same way that Python has a false.*

*S:*

*Huh. So what does none do?*

*A:*

*Ruby does not have a none*

*B:*

*nil is approximately equal to false for the purposes of this code*

*S:*

*ok i'll just say false*
*If year mod four zero put 1*
*else put nil, end end*
*can i ask like what's a leap year again?*

*B:*

*so leap year is when you have the extra day in february and there are some weird rules for when it happens that you can see happening in the is leap year function*

*S:*

*ok I don't understand why they're modding it by 400*

*A:*

*so a leap year occurs every fourth year unless its a hundredth years unless thats also a four hundredth year*
*for example in 1900 there wasn't a leap year even though its divisible by four, but in 2000 there was a leap year.*

*S:*

*ok*

**Context**: Lines 22-32, File 2
**Summary**: Student asks about null/nil/None. Student asks for clarification of what a leap year is.
**Conclusions**: Student is not very familiar with the concept the verbal prompt at beginning of question asked about (leap years).

*[19:01 - 20:57]:*
*S:*

*huh weird [laughs] ok um*
*check month*
*if month equals 13, month equals one*
*year equals year plus one, else if [pause]*
*does it just automatically return, like, true?*
*does that make sense?*
*because it just says if all these things and then doesn't say anything*

*B:*

*right so you might have noticed that i guess puts is the equivalent of return in ruby and if you notice what is this function doing within the ifs*
*it's just listing cases*

*B:*

*or the if is checking if the member variable month is equal to thirteen then below it it is saying...?*

*S:*

*oh end?*

*B:*

*or is saying at month equals one so that is an assignment*

*S:*

*OH ok i see*
*ok that makes sense*
*ok so*
*if month is thirteen it sets it equal to one and then it sets year to year plus one*
*that makes ok so its just like starting over a new year*
*month is zero set month equal to twelve*
*year minus one*

**Context**: Lines 34 - 42, File 2
**Summary**: Student asks about returning values and assignments.
**Conclusions**: Student is not comfortable with classes and class member variables.

*[20:57]:*
*S:*

*so what am I - what am I supposed to be checking for?*

*B:*

*checking if leap years are supported in the entire class*

**Context**:
**Summary**: Student asks interviewers to tell her what she is supposed to be checking for.
**Conclusions**: Student has forgotten what the initial question is in the ~7 minutes she has been working.

## 1.4.4 ACBE3Q1

All line numbers refer to the code of Question 1.1 found in the Appendix.

*[1:55-1:58]:*
*S:*

*Alright so, its go a main, so that gonna start.*

**Context:** Lines 18,12
**Summary:**

The student starts at the beginning of execution.

**Conclusion:**

The initial strategy of this student is to look for main, and then as we'll see later also continue in execution order.

*[1:58-2:12]*
*S:*

*So thats going to print whatever the result of function 3 on 1,2,3, and 4 some array. So lets see, function 3 takes a list and returns whatever function 2 does called with the list and some argument 4.*

**Context:** Lines 13, 15-16
**Summary:**

The student continues down the execution stack.

**Conclusion:**

The student's strategy for this problem is to follow the execution of the program.

*[2:12-2:22]*
*S:*

*Function 2 returns function 1 with the same two arguments already passed to it, and function 4, the result of...*

**Context:** Lines 0-1
**Summary:**

The student continues with execution is order, but is confused when func4 isn't being passed any arguments.

**Conclusion:**

The student does not realize that functions can be passed as arguments.

*[2:25]*
*S:*

*Which doesn't have any implied arguments, thats interesting. Umm,*

*[2:30]*
*[2:35] thats, odd,*

**Context:** Line 3

**Summary:**

The student is looking for a reason why func4 has no arguments on line 1, one possible reason is that in python functions can have implied arguments, i.e.

```
def func4(a=1,b=2):
        return a * b
```

note that in python, func4 would still indicate the function, and the syntax func4() would still be required to call the function (in this case returning 1*2 or 2).

**Conclusion:**

The students initial reaction is to assume that func4 is being called without any arguments. Note that in this code, main is also called without any arguments but still requires parentheses, so the student should be able to understand from the code that parentheses are required even on functions with no arguments. Without looking at func1, the student is unable to determine the semantics pertaining to func4 in the argument list of func1.

*[2:36-2:50]*
*S:*

*lets see, so its calling functi-,ooh, it calling function 1.*
*There we go. Uhh, yes, so its calling function with a list a number, the array and 4, and the function 4 as sort of a multiplier.*
*A function to apply.*

**Context:** Line 6,9

**Summary:**

The student identifies that "f" is being passed as an argument, and then called.

**Conclusion:**

The context contained in func1 is enough for the student to correctly identify the semantics for func4.

*[2:50-3:17]*
*S:*

*Alright, so function 1 is doing the actual work here.*
*Umm, see, it starts with some accumulator 0, iterates across the items in the list,*
*list and plus equals that function 4 applied to i being the item from the list,*
*and that number 4 included from function 3.*
*So its essentially going to sum the list multiplied by 4.*

**Context:** Line 6-10

**Summary:**

The student describes what specifically func1 is doing, then generalizes what func1 is doing in the context of the arguments being passed from func3.

**Conclusion:**

Following execution still allows the student to have a high level idea of what some of the functions do. After one small problem, the student is able to quickly finish the problem.

### 1.5 Figures

Fig 1. ACBE1Q3

```
$months31 = [1,3,5,7,8,10,12]
$months30 = [4,6,9,11]

class OurDate
  attr_accessor :year
  attr_accessor :month
  attr_accessor :day

  def initialize(year, month, day)
    @year = year
    @month = month
    @day = day
  end

  def is_equal?( d )
    puts @year == d.year and
      @month == d.month and
      @day = d.day
  end

  def is_leap_year?
    if @year % 400 == 0
      puts 1
    elsif @year % 100 == 0
      puts nil
    elsif @year % 4 == 0
      puts 1
```

Fig 2. ACBE4Q1

Question 1.1

```
def func2(list, num):
    return func1(list, num, func4)

def func4(a, b):
    return a * b

def func1(list, num, f):
    acc = 0
    for i in list:
        acc += f(i, num)
    return acc

def main():
  print(func3([1,2,3,4]))

def func3(list):
  return func2(list, 4)

main()
```

func3([1, 2, 3, 4])

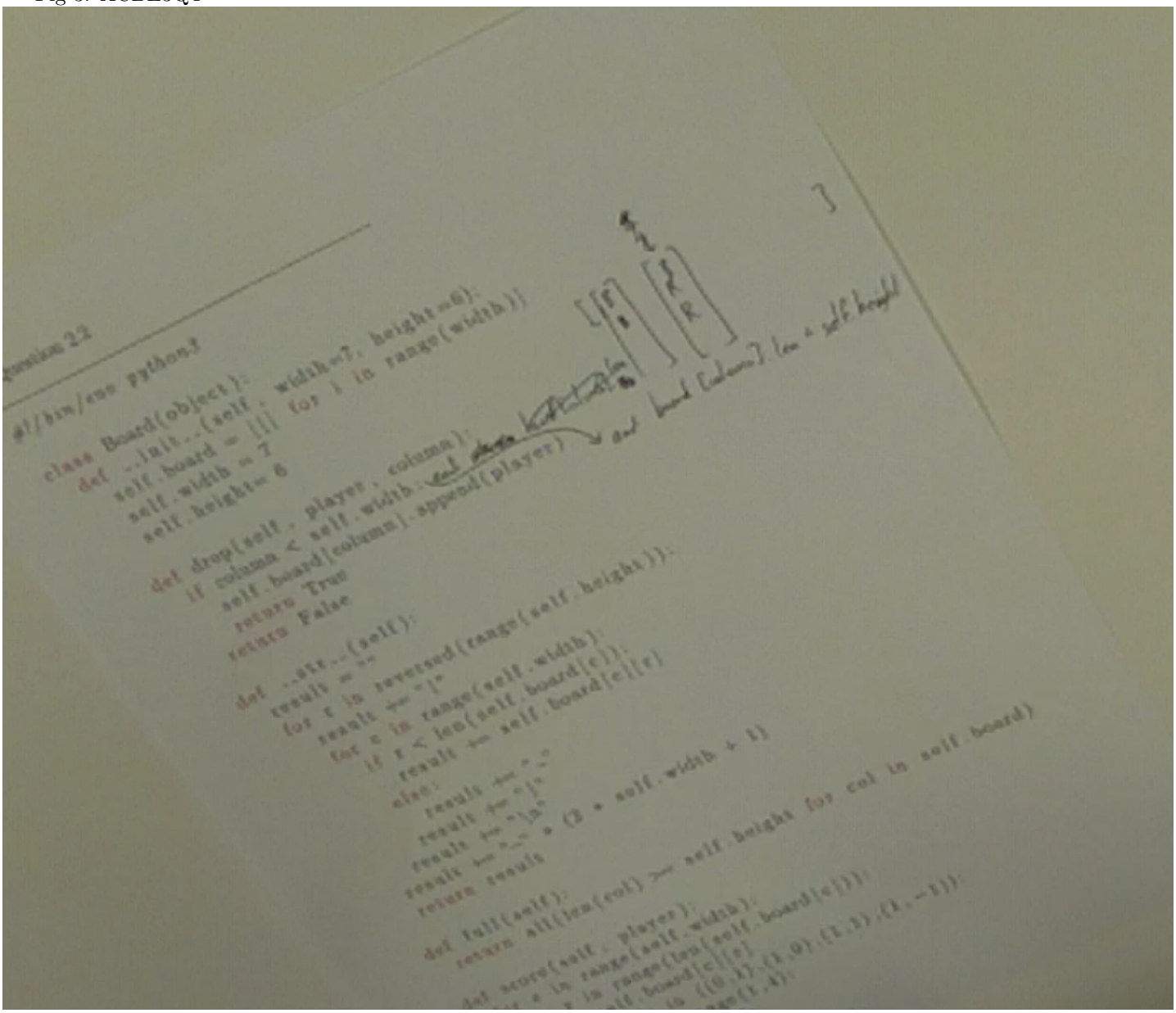
func2([1,2,3,4], 4)

func1([1,2,3,4], 4, *)

| | |
|---|---|
| acc = 0 | 0 |
| += 1 * 4 | 4 |
| += 2 * 4 | 12 |
| += 3 * 4 | 24 |
| += 4 * 4 | 40 |

"40"

Fig 3. ACBE3Q4



# 2 Appendix

## 2.1 Interview Question 1

Code given to participants did not include line numbers.
Code given to ACBE1, ACBE2, and ACBE3 was in color. Code given to ACBE4 was in grayscale.
Verbal prompt was given before handing code to participant:
"For this question we would like you to familiarize yourself with some Python code. Please explain to us what you think this code does."

---

```
0   def func2(list , num):
1     return func1(list , num, func4)
2
3   def func4(a, b):
4     return a * b
5
6   def func1(list , num, f):
7     acc = 0
8     for i in list:
9         acc += f(i, num)
10    return acc
11
12  def main():
13    print(func3([1,2,3,4]))
14
15  def func3(list):
16    return func2(list , 4)
17
18  main()
```

## 2.2 Interview Question 2

Code given to participants did not include line numbers.
Code given to ACBE1, ACBE2, and ACBE3 was in color. Code given to ACBE4 was in grayscale.
Verbal prompt was given before handing code to participant:
"For this question, we would like you to again familiarize yourself with some Python code. Please explain to us what you think this code does. "

---

```
0   def function50(i, L):
1     return L[i+2]
2
3   def function37(L):
4     return [L[-1]]+L
5
6   def function52(i):
7     return function4() * i
8
9   def function1(j, k):
10    return (j + k) * function52(1)
11
12  def function4():
13    return 3
14
15  def function188(L):
16    return function37(L)+[function50(2, L)]
17
18  def function0():
19    return function188([1,2,3,4,5,6,7,8,9])[function1(0,1)]
20
21  x = function0()
22  print x
```

## 2.3 Interview Question 3

Code given to participants did not include line numbers.
Code given to ACBE1, ACBE2, and ACBE3 was in color. Code given to ACBE4 was in grayscale. Code given to ACBE1 and ACBE2 had double equals signs that were joined together. Code given to ACBE3 and ACBE4 had spaces between the equals signs.
Verbal prompt was given before handing code to participant:
"For this question we would like to have you look at some code in the programming language Ruby. This is the scenario: A coworker recently left on vacation and left two files behind. One of those files is a Date class and your boss wasn't sure if the coworker had finished including leap year support. Your boss would like you to make sure it is supported."

---

File 1:

```
0 #!/usr/bin/ruby
1
2 load "ourdate.rb"
3
4 d = OurDate.new(2011,1,4)
5 print "#{d.what_day}"
6 print "We started writing this file today.\n"
7 d.forward_time(365)
8 print "We are almost done now.\n"
9 print "#{d.what_day}"
```

File 2

```
0   #!/usr/bin/env ruby
1
2   $months31 = [1,3,5,7,8,10,12]
3   $months30 = [4,6,9,11]
4
5   class OurDate
6     attr_accessor :year
7     attr_accessor :month
8     attr_accessor :day
9
10    def initialize(year, month, day)
11      @year = year
12      @month = month
13      @day = day
14    end
15
16    def is_equal?( d )
17      puts @year = = d.year and
18        @month = = d.month and
19        @day = d.day
20    end
21
22    def is_leap_year?
23      if @year % 400 = = 0
24        puts true
25      elsif @year % 100 = = 0
26        puts false
27      elsif @year % 4 = = 0
28        puts true
29      else
30        puts false
31      end
32    end
```

```
33
34      def check_month
35        if @month == 13
36          @month = 1
37          @year = @year + 1
38        elsif @month == 0
39          @month = 12
40          @year = @year - 1
41        end
42      end
43
44      def tomorrow
45        @day = @day + 1
46        if @day > 31
47          for i in $months31
48            if @month == i
49              @day = 1
50              @month = @month + 1
51              check_month
52            end
53          end
54        elsif @day > 30
55          for i in $months30
56            if @month == i
57              @day = 1
58              @month = @month + 1
59              check_month
60            end
61          end
62        elsif @day > 28 and @month == 2
63          @day = 1
64          @month = @month + 1
65          check_month
66        end
67      end
68
69      def yesterday
70        @day = @day - 1
71        if @day == 0
72          @month = @month - 1
73          check_month
74          for i in $months31
75            if @month == i
76              @day = 31
77            end
78          end
79          for i in $months30
80            if @month == i
81              @day = 30
82            end
83          end
84          if @month == 2
85            @day = 28
86          end
87        end
88      end
89
90      def forward_time(n)
91        for i in 0..n
92          tomorrow
```

```
93        end
94      end
95
96      def reverse_time(n)
97        for i in 0..n
98          yesterday
99        end
100     end
101
102     def what_day
103       puts "Today is #{month}/#{day}, #{year}!"
104     end
105   end
```

## 2.4 Interview Question 4

Code given to participant did not contain line numbers.
Code given to ACBE1, ACBE2, and ACBE3 was in color. Code given to ACBE4 was in grayscale. Code given to ACBE1 and ACBE2 had double equals signs that were joined together. Code given to ACBE3 and ACBE4 had spaces between the equals signs.
Verbal prompt was given before handing code to participant:
"For this question we would like to have you look at some code in Python. This is the scenario: You acquired a connect 4 program from a friend. However, the friend has warned you that you can put too many pieces in a column. Determine a possible fix for this bug so that you can enjoy your connect 4 program."

---

```
0   #!/bin/env python3
1
2   class Board(object):
3     def __init__(self, width=7, height=6):
4       self.board = [[] for i in range(width)]
5       self.width = 7
6       self.height= 6
7
8     def drop(self, player, column):
9       if column < len(self.board):
10        self.board[column].append(player)
11        return True
12      return False
13
14    def __str__(self):
15      result = ""
16      for r in reversed(range(self.height)):
17        result += "|"
18        for c in range(self.width):
19          if r < len(self.board[c]):
20            result += self.board[c][r]
21          else:
22            result += "␣"
23          result += "|"
24        result += "\n"
25      result += "–" * (2 * self.width + 1)
26      return result
27
28    def full(self):
29      return all(len(col) >= self.height for col in self.board)
30
31    def score(self, player):
32      for c in range(self.height):
33        for r in range(len(self.board[c])):
34          p = self.board[c][r]
35          for dc,dr in ((0,1),(1,0),(1,1),(1,-1)):
36            for i in range(1,4):
37              nc = c + i*dc
38              nr = c + i*dr
39              if nc < 0 or self.width <= nc:
40                break
41              if nr < 0 or len(self.board[nc]) <= nr:
42                break
43              if self.board[nc][nr] != p:
44                break
45            else:
46              return 1 if p == player else -1
47      return 0
48
```

```
other = {'X' : 'O', 'O' : 'X'}
player = 'X'
board = Board()

while True:
  try:
    c = int(input("%s > " % player))
  except TypeError:
    continue
  if not board.drop(player,c):
    continue
  print(board)
  if board.score(player):
    print("Player %s Wins!!!" % player)
  elif board.full():
    print("Tie")
  else:
    player = other[player]
    continue
  board = Board()
  player = 'X'
  print(board)
```