

Task 3: Evaluating Code with Conditionals

a.

```
main.py
1 def smallest(n:float, m:float) -> float: 2 usages
2     if n < m:
3         return n          # For which calls below is this statement evaluated? Neither of them
4     else:
5         return m
6
7 first = smallest( n: 3, m: 2)      # What is the value of first? 2
8 second = smallest( n: 2, m: 2)     # What is the value of second? 2 Is this a reasonable result? Why or why not?
9                                     # I would say this is not a reasonable result because n and m are the same value
10 print()
```

b.

```
main.py
1 def function2(a:int, b:int, c:int) -> int: 3 usages
2     if a > b and a > c:
3         return a - b      # In general, when will a call to this function evaluate this statement?
4     elif b > c:
5         return b + c      # When a is the largest value
6     else:
7         return 2 * c       # In general, when will a call to this function evaluate this statement?
8                                     # If neither a is the largest number or b is greater than c
9
10 answer1 = function2( a: 3, b: 2, c: 1)    # What is the value of answer1? 3-2, 1
11 answer2 = function2( a: 2, b: 3, c: 1)    # What is the value of answer2? 3+1, 4
12 answer3 = function2( a: 2, b: 1, c: 3)    # What is the value of answer3? 2*3, 6
13 print()
```

Task 4: Evaluating Code with Lists and Strings

a.

PyCharm IDE showing the main.py file. The code defines a function `checked_access` that returns the element at index `idx` if `idx` is within the bounds of the list `L`, or `None` otherwise. It also shows two calls to this function with comments explaining the expected behavior based on the index values.

```
from typing import Optional      # gain access to the Optional[X] type hint

def checked_access(L:list[int], idx:int) -> Optional[int]:  2 usages
    test = idx >= 0 and idx < len(L)    # What is the value of test on each call? Test is a boolean expression based on if idx is >=0
    if test:
        return L[idx]
    else:
        return None

first = checked_access([1, 0, 1], idx: 9)    # What is the value of first? test false, so return None
second = checked_access([1, 0, 1], idx: 2)    # What is the value of second? test true, so return length of index which is 3
print()
```

The PyCharm interface includes a toolbar, a file tree, a code editor with syntax highlighting, a navigation bar, and a bottom status bar indicating the file path, encoding, and Python version.

a.

b.

PyCharm IDE showing the main.py file. The code defines a function `length_sum` that calculates the sum of the lengths of all strings in a list `L`. It includes several conditional statements to handle different cases of the input list's length.

```
def length_sum(L:list[str]) -> int:  3 usages
    if len(L) > 2:
        result = len(L[0]) + len(L[1]) + len(L[2])    # For which call below is this statement evaluated? For the first call
    elif len(L) > 1:
        result = len(L[0]) + len(L[1])    # For which call below is this statement evaluated? The third call
    elif len(L) > 0:
        result = len(L[0])    # For which call below is this statement evaluated? The first 2 values, 7+4=11
    else:
        result = 0    # For which call below is this statement evaluated? The only term, 11

    return result

first = length_sum(["this", "is", "the", "first", "call"])
second = length_sum(["second call"])
third = length_sum(["another", "call"])
print()
```

The PyCharm interface includes a toolbar, a file tree, a code editor with syntax highlighting, a navigation bar, and a bottom status bar indicating the file path, encoding, and Python version.

b.

```
main.py
1 def surprising(L:list[str], other:str) -> list[str]: 2 usages
2     L.append(other.upper())
3     return L
4
5
6 words = ["this", "is", "confusing", "code."]
7 first = surprising(words, other="Avoid")
8 second = surprising(words, other="Such.")
9     # What is the value of words at this point? words is a modified list at this point
10    # What are the values of first and second at this point? AVOID & SUCH
11    # What happened? Because lists are mutable both AVOID and SUCH append the list making it,
12    # ["this", "is", "confusing", "code.", "AVOID", "SUCH"]
13 print()
```

The screenshot shows the PyCharm IDE interface with a Python project named "PythonProject2.1". The main editor window displays a file named "main.py" containing the provided code. The code defines a function "surprising" that takes a list and a string, appends the string's uppercase version to the list, and returns the list. The script then creates a list of words, calls "surprising" twice with different strings ("Avoid" and "Such."), and prints the final list. The output in the terminal window shows the final state of the list as having six elements: "this", "is", "confusing", "code.", "AVOID", and "SUCH". The PyCharm status bar at the bottom indicates the file is saved, the encoding is UTF-8, and the Python version is 3.13.

Worksheets

Week 2 Worksheet 1																															
1. Complete trace table for the following code.																															
def abs(n:int)→int: if n>0: return n else: return -n first = abs(9) second = abs(-4)	<table border="1"> <thead> <tr> <th>initial code</th> <th>known Bindings</th> </tr> </thead> <tbody> <tr> <td>Line 8</td><td>first = 9</td></tr> <tr> <td></td><td>call to abs(9) → 9</td></tr> <tr> <td>Line 1</td><td>n = 9</td></tr> <tr> <td></td><td>2. n > 0 → TRUE</td></tr> <tr> <td></td><td>3. return 9</td></tr> <tr> <td></td><td>4. not executed</td></tr> <tr> <td></td><td>5. "</td></tr> <tr> <td>Line 9</td><td>first = -4</td></tr> <tr> <td></td><td>call to abs(-4) → -4</td></tr> <tr> <td>Line 1</td><td>n = -4</td></tr> <tr> <td></td><td>2. n < 0 → False</td></tr> <tr> <td></td><td>3. Not executed</td></tr> <tr> <td></td><td>4. -4 > 0 → False</td></tr> <tr> <td></td><td>5. return -4</td></tr> </tbody> </table>	initial code	known Bindings	Line 8	first = 9		call to abs(9) → 9	Line 1	n = 9		2. n > 0 → TRUE		3. return 9		4. not executed		5. "	Line 9	first = -4		call to abs(-4) → -4	Line 1	n = -4		2. n < 0 → False		3. Not executed		4. -4 > 0 → False		5. return -4
initial code	known Bindings																														
Line 8	first = 9																														
	call to abs(9) → 9																														
Line 1	n = 9																														
	2. n > 0 → TRUE																														
	3. return 9																														
	4. not executed																														
	5. "																														
Line 9	first = -4																														
	call to abs(-4) → -4																														
Line 1	n = -4																														
	2. n < 0 → False																														
	3. Not executed																														
	4. -4 > 0 → False																														
	5. return -4																														
2.) Line 1 def price(age:int)→int: 2. if age < 10: / 3. return 7 4. elif age > 55: / 5. return 12 6. else: / 7. return 17 10. cost = price(19)	<table border="1"> <thead> <tr> <th>initial code</th> <th>known Bindings</th> </tr> </thead> <tbody> <tr> <td>Line 10</td><td>cost = 17</td></tr> <tr> <td></td><td>call to price(19) → 17</td></tr> <tr> <td>Line 1</td><td>age = 19</td></tr> <tr> <td></td><td>2. Not executed</td></tr> <tr> <td></td><td>3. "</td></tr> <tr> <td></td><td>4. "</td></tr> <tr> <td></td><td>5. "</td></tr> <tr> <td></td><td>6. 19 < 10 → False; 19 > 55 → False</td></tr> <tr> <td></td><td>7. return 17</td></tr> </tbody> </table>	initial code	known Bindings	Line 10	cost = 17		call to price(19) → 17	Line 1	age = 19		2. Not executed		3. "		4. "		5. "		6. 19 < 10 → False; 19 > 55 → False		7. return 17										
initial code	known Bindings																														
Line 10	cost = 17																														
	call to price(19) → 17																														
Line 1	age = 19																														
	2. Not executed																														
	3. "																														
	4. "																														
	5. "																														
	6. 19 < 10 → False; 19 > 55 → False																														
	7. return 17																														

Week 2 Worksheet 2

```
1 def abs(n:int) -> int:  
2     if n > 0:  
3         return n  
4     else:  
5         return -n
```

(a.) ~~n > 0~~ → true evaluated by line 3.

(b.) ~~n < 0~~ before evaluation of line 5.

```
2. 1. def price(age:int) -> int:  
2.     if age < 10:  
3.         return 7  
4.     elif age > 55:  
5.         return 12  
6.     else:  
7.         return 17
```

(a.) age < 10 → true by evaluation of line 3

(b.) age > 55 → true by evaluation of line 5

(c.) age < 10, age > 55 both true by evaluation of line 7.

Week 2 worksheet 3

1.) $L = [1, 2, 3]$

$$L[1] + L[2]$$

$$2 + 3$$

$$5$$

2.) $L = [1, 2, 3]$

$$L[L[0]]$$

$$L[1]$$

$$2$$

3.) $L = [4, 2, 9, 7]$

$$L[-3] + L[0]$$

$$2 + 4$$

$$6$$

4.) $s = "hello"$

len(s)

$$5$$

5.) $s = "hello\nthere",$

$$s[2] + s[5] + s[-3]$$

$$l + \backslash n + r$$

$$'l\nr'$$

Weeks 2 Worksheet - 4

1 def questionable(argsL: list[int], idx: int, value: int) → bool:

2 if $idx \geq 0$ and $idx < len(argsL)$:

$argsL[idx] = value$

return True

else:

return False

Known Bindings

L = [4, 2, 5, 3, 1, 9]

L = [4, 2, 5, 3, 1, 9]

Line 10

L = [4, 2, 5, 10, 1, 9]

first = questionable(L, 3, 10)

second = questionable(L, 2, 0)

L[1]S

L = [4, 2, 5, 3, 1, 9]

L = [4, 2, 5, 10, 1, 9]

L = [4, 2, 0, 10, 1, 9]

Line 11

L = [4, 2, 0, 10, 1, 9]

Initial code

call to first = questionable(L, 3, 10) → L = [4, 2, 5, 10, 1, 9]

1 argsL = [4, 2, 5, 3, 1, 9], idx = 3, value = 10

2 idx ≥ 0 → True idx < len(argsL) → True

3 argsL[3] = 10

4 return True

9 L = [4, 2, 5, 10, 1, 9]

call to questionable(L, 2, 0) → L = [4, 2, 0, 10, 1, 9]

1 argsL = [4, 2, 0, 10, 1, 9], idx = 2, value = 0

2 idx ≥ 0 → True idx < len(argsL) → True

3 argsL[2] = 0

4 return True

9 L = [4, 2, 0, 10, 1, 9]

Week 2 Worksheet 5

1.)

```

1 listOne = []
2 listOne.append(1)
3 listOne.append(3)
4 listOne.append(9)
5
6 listOne + [4]
    
```

Lists

```

L = []
L = [1]
L = [1, 3]
L = [1, 3, 9]
L = [1, 3, 9]
L = [1, 3, 9, 4]
    
```

initial code

call to $listOne + [4] \Rightarrow listOne = [1, 3, 9, 4]$

```

1 listOne = []
2 listOne = []
3 listOne = [1, 3]
4 listOne = [1, 3, 9]
5 listOne = [1, 3, 9, 4]
    
```

known Bindings

line 1	listOne = []
line 2	listOne = [1]
line 3	listOne = [1, 3]
line 4	listOne = [1, 3, 9]
line 6	listOne = [1, 3, 9, 4]

2.)

```

1 listOne = [4, 2, 5]
2 listTwo = [8, 9]
3 first = listOne + listTwo
4 second = listOne.append(listTwo)
5 third = listOne.extend(listTwo)
    
```

Lists

```

listOne = [4, 2, 5] listTwo = [8, 9]
listNew = [4, 2, 5, 8, 9]
listOne = [4, 2, 5, 8, 9]
listOne = [4, 2, 5, 8, 9]
    
```

initial code

call to $first = listOne + listTwo \Rightarrow L = [4, 2, 5, 8, 9]$

```

1 listOne = [4, 2, 5]
2 listTwo = [8, 9]
3 L = [4, 2, 5, 8, 9]
    
```

call to $second = listOne.append(listTwo) \Rightarrow L = [4, 2, 5, 8, 9]$

```

1 listOne = [4, 2, 5]
2 listTwo = [8, 9]
4 listOne = [4, 2, 5, 8, 9]
    
```

call to third = listOne . extend (listTwo) \Rightarrow listOne = [4, 2, 5, [8, 9]]

1 listOne = [4, 2, 5]

2 listTwo = [8, 9]

3 listOne = [4, 2, 5, [8, 9]]

Known Bindings

Line 1 | listOne = [4, 2, 5]

Line 2 | listTwo = [8, 9]

Line 3 | L = [4, 2, 5, 8, 9]

Line 4 | listOne = [4, 2, 5, 8, 9]

Line 5 | listOne = [4, 2, 5, [8, 9]]

Week 2 Worksheet 5

3) $\text{listOne} = [4, 2, 5]$

1 other = listOne

3 listOne.append(0)

initial code

call to listOne.append(0) $\rightarrow \text{listOne} = [4, 2, 5, 0]$

1 listOne = [4, 2, 5]

2 other = listOne

3 listOne = [4, 2, 5, 0]

Lists

listOne = [4, 2, 5]

listOne = [4, 2, 5, 0]

Known Bindings

Line 1 | $\text{listOne} = [4, 2, 5]$

Line 2 | $\text{listOne} = [4, 2, 5, 0]$, other = [4, 2, 5]

Line 3 | $\text{listOne} = [4, 2, 5, 0]$

Week 2 Worksheet 6

1.)

1 sentence = "This is a poorly formatted sentence."

2 words = sentence.split()

3 new = ''.join(words)

lists

words = ["This", "is", "a", "poorly", "formatted", "sentence"]

new

initial code

call to new = ''.join(words) \rightarrow "This is a poorly formatted sentence!"

1. sentence = "This is a poorly formatted Sentence."

2. words = ["This", "is", "a", "poorly", "formatted", "sentence"]

3. new = "This is a poorly formatted sentence."

known bindings

Line 1 sentence = "This..."

Line 2 words = ["This", etc.]

Line 3 new = "This..."

2.)

1 def capitalize(s: str) \rightarrow str:

2 chars = [i for i]

3 chars[0] = chars[0].upper()

4 return "", join(chars)

7 word = capitalize("hello")

initial code

call to word = capitalize("Hello") \rightarrow Hello

1 s = "Hello"

2 chars = [h, e, l, l, o]

3 chars = [H, e, l, l, o]

4. return "Hello"

lists

1: str(s) = [h, e, l, l, o]

1: str(s) = [H, e, l, l, o]

known bindings

Line 7 "Hello"