Quiz 01 - Practice

COMP 110: Introduction to Programming
Spring 2024

Thursday 15, 2024

Name: _____

9-digit PID: _____

Do not begin until given permission.

Honor Code: I have neither given nor received any unauthorized aid on this quiz.

Signed: _____

Question 1: Multiple Choice Completely fill in the bubble next to your answer using a pencil. Each question should have exactly one filled-in bubble.

1.1. The following string is an example of an format string:

```
1 "{1 + 1}"
```

- ◯ `True`
- ● `False`

1.2. What does the following string evaluate to?

```
1 "\\\\"
```

- ◯ \
- ● \\
- ◯ \\\
- ◯ \\\\

1.3. What is the printed output of the following `print` function call?

```
1 print(f"C{'OM'}P{100 + 10}")
```

- ◯ `fCOMP10010`
- ● `COMP110`
- ◯ `C'OM'P100 + 10`
- ◯ `Error: Invalid Syntax`

1.4. What does the `chr` function do in the following example:

```
1 chr(129312)
```

- ● `Converts an int representation into a string character`
- ◯ `Converts a string character into an int representation`
- ◯ `Chars a number by burning it just a little`
- ◯ `Error: This function is not built-in to Python`

1.5. What is the *type* and *evaluation* of this expression in Python?

```
1 "ABCD" < "ABCZ"
```

- ◯ `False`
- ● `True`

1.6. Hexidecimal is base-16, binary is base-2, and decimal is base-10.

- ◯ `False`
- ● `True`

1.7. Which operator has the highest precedence in an expression?

- ◯ or
- ◯ >
- ● +
- ◯ and
- ◯ not

1.8. What is the evaluation of the following expression:

```
1 1 > 0 or "B" > "A"
```

- ◯ False
- ● True

1.9. What is the evaluation of the following expression:

```
1 "A" == "B" and "B" == "C"
```

- ● False
- ◯ True

1.10. A Tuple can hold 0, 1, or more values:

- ◯ False
- ● True

1.11. What is the evaluation of the following expression:

```
1 (110,210,301)[1 + 1]
```

- ◯ 0
- ◯ 1
- ◯ 110
- ◯ 210
- ● 301
- ◯ Error

1.12. What is the evaluation of the following Python expression?

```
1 not True or True
```

- ◯ `False`
- ● `True`

1.13. What is the evaluation of the following expression?

```
1 (1,) + (1, 0)
```

- ◯ `(1, 0)`
- ◯ `(2, 0)`
- ● `(1, 1, 0)`
- ◯ `Error`

1.14. Which of the following are required in a recursive function that does not infinitely recur?

- ◯ A base case without a recursive function call
- ◯ Arguments changing in recursive case
- ◯ Recursive cases make progress toward the base case
- ● All of the above

1.15. Which of the following is a valid function call to the following function signature?

```
1 def a_func(x: int, y: int = 0)
      -> int:
2   ...
```

- ◯ A. `a_func()`
- ◯ B. `a_func(1)`
- ◯ C. `a_func(1, 2)`
- ● B and C
- ◯ None of the above

1.16. What type of error occurs when recursion appears to infinitely

- ◯ `Name Error`
- ◯ `Index Error`
- ● `Stack Overflow Error`
- ◯ `Syntax Error`

1.17. What will the following Python expression evaluate to?

```
1 1 + True
```

- ◯ `True`
- ● `2`
- ◯ `1`
- ◯ `False`

1.18. What is the following statement declaring?

```
1 PI: float = 3.14
```

- ◯ Global Named Constant
- ◯ Local Named Constant
- ● Either of the above, depending on where it is declared
- ◯ None of the above

1.19. Consider the following function declaration:

```
1 def a_func(x: int, y: int = 0)
      -> int:
2   ...
```

Which of the following are valid ways of calling the function?

- ◯ A. `a_func(x=1, y=2)`
- ◯ B. `a_func(x=1)`
- ◯ C. `a_func(1, 2)`
- ◯ A and B
- ● A, B, and C
- ◯ None of the above

1.20. What does the built-in `id` function evaluate to when called?

- ◯ The part of a computer's brain an object is in.
- ● The ID, which is the memory address, of the argument it is given.
- ◯ The *identity* of its argument, e.g. the argument itself

Question 2: Respond to the following questions

Consider the following code listing:

```
def eight_ball(choice: int) -> str:
  """Returns an 8-ball response."""
  if choice <= 0:
    return "Unlikely."
  else:
    if choice > 0:
      return "It is certain."
    else:
      return "Ask again later."
```

2.1. Write a function call expression to the `eight_ball` function that evaluates to `"It is certain."`

Solution: `eight_ball(1)` or `eight_ball(choice=1)` or any argument value greater than 1

2.2. Write a function call expression to the `eight_ball` function that evaluates to `"Unlikely."`

Solution: `eight_ball(0)` or `eight_ball(choice=0)` or any argument value less than 0

2.3. Write a function call expression to the `eight_ball` function that evaluates to `"Ask again later."`

Solution: This code is unreachable and no function call can be made, as written, to result in "Ask again later."

2.4. What value and type does the following expression evaluate to: `3 + 4 == 6`

Solution: False, bool

2.5. What value and type does the following expression evaluate to?

```
((True and False) or (False or True)) != False
```

Solution: True

2.6. What value and type does the following expression evaluate to? (This is a notably obtuse expression, but breaking it down and simplifying it will help you reinforce your understanding of expressions with subscription notation.)

```
(1, 2, 3)[(0, 1, 2)[1 - int("012"[1])]]
```

Solution: 1

Question 3: Memory Diagram Trace a memory diagram of the following code listing and then answer the sub-questions. You do not need to diagram the sub-questions.

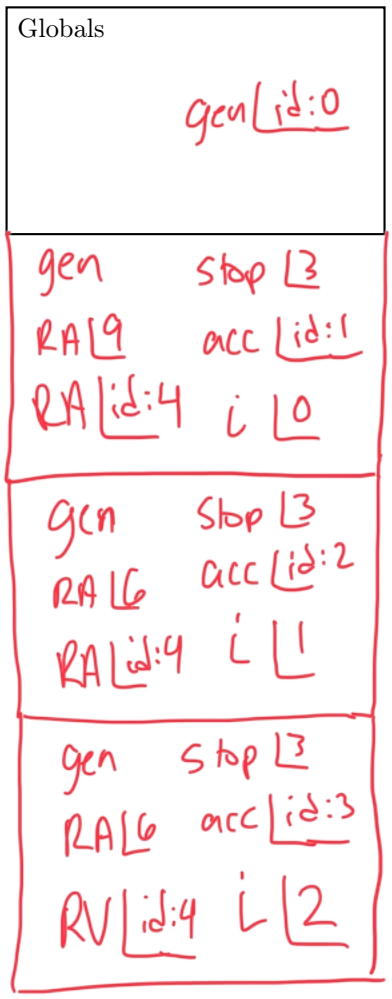
```
1  def gen(stop: int, acc: tuple[int, ...] = (), i: int = 0) -> tuple[int, ...]:
2    """Generate a tuple from i to stop."""
3    if i >= stop - 1:
4      return acc + (i,)
5    else:
6      return gen(stop, acc + (i,), i + 1)
7
8
9  print(gen(3))
```

* Each tuple concatenation produces a new tuple object on the heap

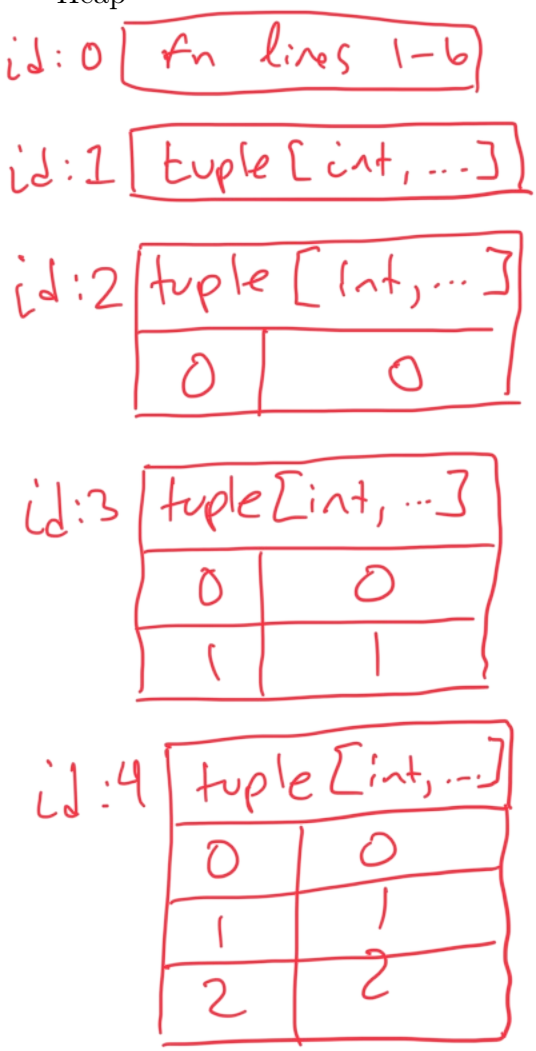
Output

Solution: (0, 1, 2)

Stack



Heap

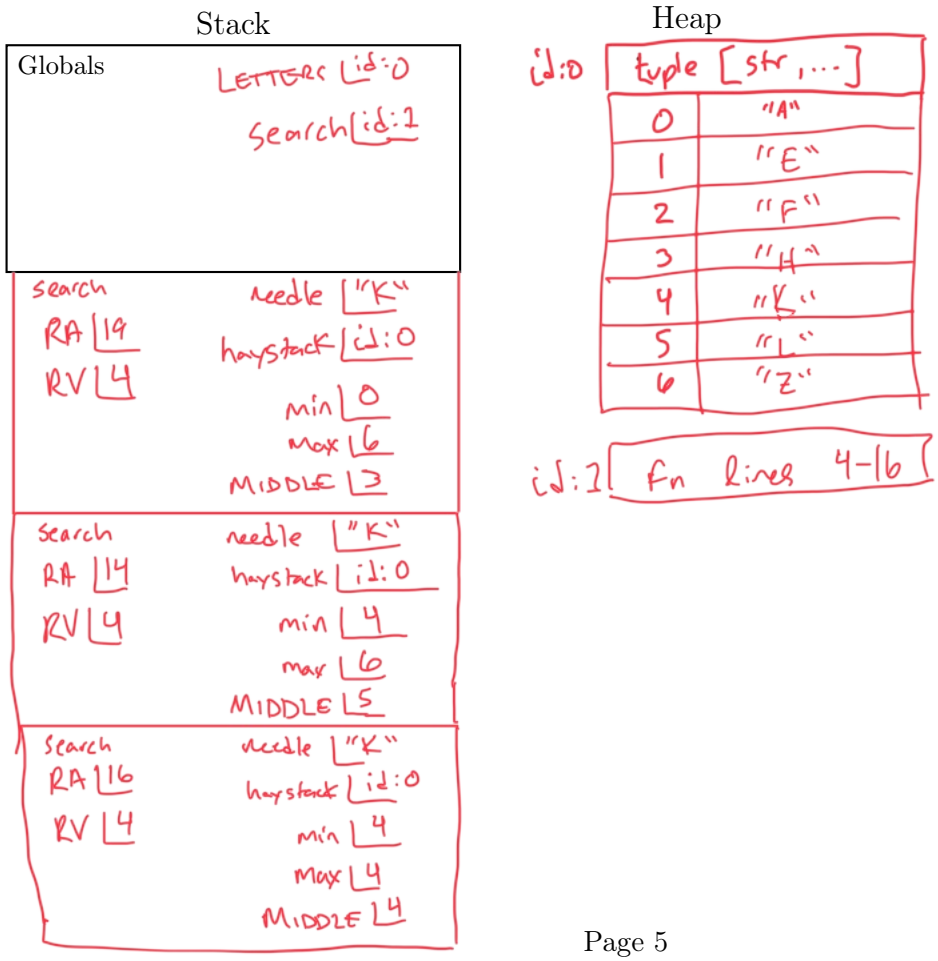


Question 4: Memory Diagram Trace a memory diagram of the following code listing and then answer the sub-questions. You do not need to diagram the sub-questions.

```
1  LETTERS: tuple[str, ...] = ("A", "E", "F", "H", "K", "L", "Z")
2
3
4  def search(needle:str, haystack:tuple[str, ...], min: int, max: int) -> int:
5    """Find the index of a needle in a sorted haystack, or -1 if not found."""
6    if min > max:
7      return -1
8    else:
9      MIDDLE: int = ((max - min) // 2) + min
10     print(f"Guess: {MIDDLE}")
11     if needle == haystack[MIDDLE]:
12       return MIDDLE
13     elif needle > haystack[MIDDLE]:
14       return search(needle, haystack, MIDDLE + 1, max)
15     else:
16       return search(needle, haystack, min, MIDDLE - 1)
17
18
19 print(search(needle="K", haystack=LETTERS, min=0, max=len(LETTERS) - 1))
```

Output (You can write successive lines beside one another separated by a //)

Solution: GUESS: 3 // GUESS: 5 // GUESS: 4 // 4



4.1. Knowing that the `haystack` `tuple` is *sorted* in ascending order, describe the general strategy this algorithm takes for finding the index of the `needle` parameter in the `haystack`.

Solution: The algorithm computes the middle index of the remaining range of indices the needle may be in. It "guesses" this middle index.

If the needle is found at the middle, great! Return this index. Otherwise, if the guess was too low, the recursive case narrows the search range by half by setting the minimum possible index to be one greater than the middle. Finally, if the guess was too high, this recursive case cuts the search range in half by setting the maximum possible index to be one less than the middle.

More simply, each recursive call narrows the search range by half by guessing in the middle of the remaining indices. If the point is reached of there being no more indices to guess, -1 is returned to indicate the needle was not found in the haystack.

4.2. On the previous code listing, what lines do you find the `return` statements of the *base cases* of the `search` function?

Solution: Lines 7 and 12

4.3. On the previous code listing, what lines do you find the `return` statements of the *recursive cases* of the `search` function?

Solution: Lines 14 and 16

4.4. One of the conditions for writing a recursive function that is not infinite is that the recursive cases make progress toward the base case(s). How do the recursive cases make progress toward the base case resulting in `-1`?

Solution: The base case states that min must be greater than max.

This means that either min must be increasing toward max, or max must be decreasing toward min, or both.

The first recursive case increases min while holding max the same. The second recursive case decreases max while holding min the same. Therefore, each recursive case is making progress toward this base case.

Question 5: Memory Diagram Trace a memory diagram of the following code listing and then answer the sub-questions. You do not need to diagram the sub-questions.

```
1  def fib(n: int) -> int:
2    """Compute the fibonacci of n"""
3    print(f"fib({n})")
4    if n == 0 or n == 1:
5      return n
6    else:
7      N1: int = fib(n - 1)
8      N2: int = fib(n - 2)
9      return N1 + N2
10
11 print(fib(3))
```

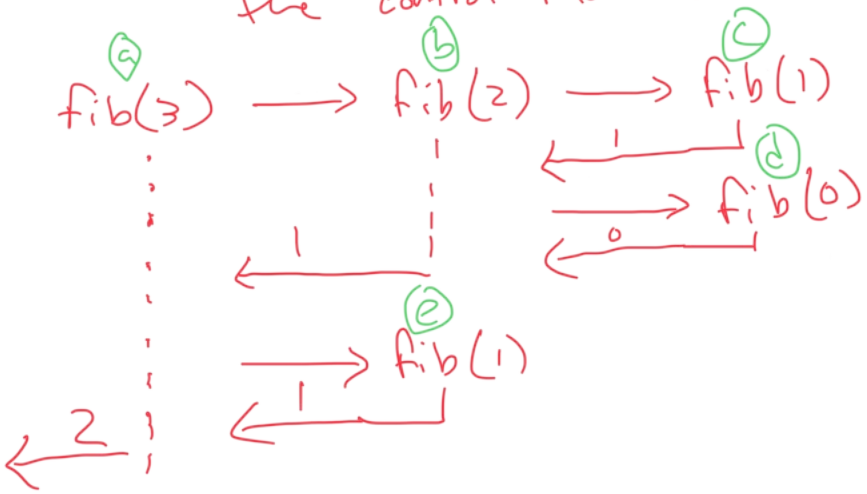
Output

Solution: fib(3) // fib(2) // fib(1) // fib(0) // fib(1) // 2

Stack

Heap

Not needed in diagram, but for explanation purposes, the control flow is:



This page intentionally left blank. Do not remove from quiz packet.