CSCI 150: Practice Exam 2 Solutions

1. Eustace is tasked with writing a function that takes a list of numbers as input and returns `True` if every even number in the list is adjacent to an odd number, and `False` otherwise. He writes the following incorrect code:

```
from typing import *

def adjacent(nums: List[int]) -> bool:
  for i in nums:
    if nums[i] % 2 == 0:
      if nums[i - 1] % 2 == 1 or nums[i + 2] % 2 == 1:
        return True
      else:
        return False
```

Describe three things that are wrong with his code and how you would fix it.

(a) The for loop needs to loop over the *indices* of `nums`, not the elements.

(b) The `return True` should be outside the loop; we won't know if everything in the list satisfies the condition until the very end.

(c) The element next to `nums[i]` is `nums[i + 1]`, not `nums[i + 2]`.

(d) If we start at the first index, `nums[i-1]` will yield an "index out of bounds" error.

There are many ways to write a correct version of `adjacent`; here is one:

```
from typing import *

def adjacent(nums: List[int]) -> bool:
  for i in range(len(nums)):
    if nums[i] % 2 == 0:
      hasodd: bool = False
      if i > 0 and nums[i-1] % 2 == 1:
        hasodd = True
      if i < len(nums) - 1 and nums[i+1] % 2 == 1:
        hasodd = True
        if not hasodd:
            return False
    return True
```

2. What is the value of `n` after execution of the following Python program?

```
t = "WHEREINTHEWORLDIS"
r = "CarmenSandiego"
p = t.find("R") * 4 - 1
f = r[-3:]
n = t[p:p + 2] + f + "N"
n.lower()
```

After execution of this program, `n = "ORegoN"`.

3. Consider the function below, which prompts the user for two numbers and then prints their sum.

```
def get_sum():
    valid = False
    while not valid:
      value1 = input("Enter a number:")
      if value1.isdigit() or value1[0] == '-' and value1[1:].isdigit():
        valid = True
        value1 = int(value1)
      else:
        print("Not a number, try again!")

    valid = False
    while not valid:
      value2 = input("Enter another number:")
      if value2.isdigit() or value2[0] == '-' and value2[1:].isdigit():
        valid = True
        value2 = int(value2)
      else:
        print("Not a number, try again!")

    return value1 + value2
```

Although it works, it has a lot of duplicated code.

First, write a function called `input_integer(prompt)` that abstracts the process of acquiring a single valid integer from the user (which happens twice in the above code). Be sure to include comments explaining the function's input(s), output(s), and description. (On the next page, you will use `input_integer` to simplify `get_sum`.)

```
def input_integer(prompt: str) -> int:
  valid = False
  while not valid:
    value = input(prompt)
    if value.isdigit() or value[0] == '-' and value[1:].isdigit():
      valid = True
      value = int(value)
    else:
      print("Not a number, try again!")

  return value
```

Now, use `input_integer` to simplify the definition of `get_sum`. Your definition of `get_sum` below should have exactly the same behavior as the original `get_sum` on the previous page.

```
def get_sum() -> int:
  value1 = input_integer("Enter a number:")
  value2 = input_integer("Enter another number:")
  return value1 + value2
```

4. We discussed the Collatz conjecture earlier in the semester. Here is a similar process for an arbitrary positive integer n:

   - If n is a multiple of three, divide it by three.
   - Otherwise, double it and add one.

   For example, starting with $n = 10$, one gets the sequence 10, 21, 7, 15, 5, *etc.*

   Using a while loop, write a function in Python that accepts a number n as a parameter, and repeats the process above until either $n = 1$ or until 100 steps are reached. The function should return how many steps the process took to reach 1, or return -1 if the process went on for 100 steps. **This function should not `input` anything from the user nor `print` anything.**

```
def splaz(n: int) -> int:
  steps: int = 0
  while n != 1 and steps < 100:
    if n % 3 == 0:
      n //= 3
    else:
      n = 2*n + 1

    steps += 1

  if steps == 100:
    return -1
  else:
    return steps
```

5. On the next page, trace the execution of the following Python program, showing the function stack, local variables of each function call, the return value of each function call, and any printed output.

```
def pheasant(n: int) -> int:
  return 5

def fish(p: int) -> int:
  p *= 3
  return p + 1

def frog(p: int) -> int:
  p += 5
  if p < 7:
    return fish(p)
  else:
    return pheasant(p)

def main():
  print(frog(1))
  print(frog(2))

main()
```

Scratch	**Stack**

Printed output

Appendix: some common string methods

```
S.count(sub: str) -> int

    Return the number of non-overlapping occurrences of sub-
    string sub in string S.

S.find(sub: str) -> int

    Return the smallest index in S where substring sub is found.
    Return -1 if sub is not found.

S.replace(old: str, new: str) -> str

    Return a copy of string S with all occurrences of substring
    old replaced by new.

S.isdigit() -> bool

    Return True if all characters in S are digits
    and there is at least one character in S, False otherwise.

S.upper() -> str

    Return a copy of the string S converted to uppercase.

S.lower() -> str

    Return a copy of the string S converted to lowercase.
```