Name: ______________________________

Please answer the following questions within the space provided on the online template. Format your solutions as well as you are able within the online text editor. While you are not required to document your code here, comments may help me to understand what you were trying to do and thus increase the likelihood of partial credit should something go wrong. If you get entirely stuck somewhere, explain in words as much as possible what you would try.

Each question clearly shows the number of points available and should serve as a rough metric to how much time you should expect to spend on each problem. You can assume that you can import any of the common libraries we have used throughout the semester thus far.

The exam is partially open, and thus you are free to utilize:

- The text
- Your notes
- Class slides
- Any past work you have done as part of sections, problem sets, or projects, provided it has been uploaded, and you access it through GitHub.

While you are allowed to use a computer for ease of typing and accessing the above resources, you are prohibited from accessing and using any editor or terminal to run your code. Visual Studio Code or any similar editor should never be open on your system during this exam. Additionally, you are prohibited from accessing outside internet resources beyond the webpages described above. *Your work must be your own on this exam, and under no conditions should you discuss the exam or ask questions to anyone but myself.* Failure to abide by these rules will be considered a breach of Willamette's Honor Code and will result in penalties as set forth by Willamette's academic honesty policy.

Please sign and date the below lines to indicate that you have read and understand these instructions and agree to abide by them. *Failure to abide by the rules will result in a 0 on the test.* Good luck!!

______________________________ ______________________________

Signature Date

(6) 1. **Evaluating Python Expressions:** For each of the below expressions, write out **both** the value of the expression and what type of object that value is (`int`, `float`, `bool`, etc). If the expression results in an error, state as much and indicate precisely where it went wrong. You can assume that the constant `HEXCHARS` has already been defined earlier in the code as `HEXCHARS = "0123456789ABCDEF"`, and has the internal representation shown below, where I've labeled both positive and negative indices.

HEXCHARS

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

(a)
```
8 * 5 + 7 % 4 - 9 // 3 ** 2 + 0 / 6
```

Solution: `42.0`, a `float`

(b)
```
not (4 < 9) or str(5*2) != str(5)+str(2)
```

Solution: `True`, a `bool`

(c)
```
HEXCHARS[-5:] + HEXCHARS[-6::-3]
```

Solution: `"BCDEFA741"`, a `str`

(12) 2. **Program Tracing:** What output would the following program produce?

```
def mystery(w):

    def enigma(s, k):
        return s[4 - k % 3]

    s = ""
    for i in range(4):
        if i % 3 == 0:
            s += w[len(w) - 1]
        else:
            s += w[0]
        s += enigma(w, i)
    return s

if __name__ == '__main__':
    print(mystery("abcdefgh"))
```

Show your work and explain your reasoning for full credit.

Solution: The final output value would be `"headache"`. Remember that you need to compute the remainder in the `enigma` function before the subtraction!

(14) 3. **Fundamental Python:** Write a function that takes a positive integer N as an argument and then calculates the first sum of the first N odd integers, returning the solution. For example, if N is 4, then your function should return the value 16, which is $1+3+5+7$.

Solution: One possible solution:

```python
def sum_odds(N):
    total = 0
    for i in range(1, 2*N, 2):
        total += i
    return total
```

or, alternatively

```python
def sum_odds(N):
    num = 0
    count = 0
    total = 0
    while count < N:
        if num % 2 == 1:
            total += num
            count += 1
        num += 1
    return total
```

(18) 4. Many words in the English language have doubled up letter combinations. For example, the word `committee` has three such pairings: the `m`, the `t`, and the `e`. Your task in this problem is to write a function called `remove_dups` that takes a string as an argument and returns a new string where each adjacent duplicate letter has been replaced with a single copy of that letter. So in our above example, calling `remove_dups("committee")` would just return `"comite"`. Note that it is only *adjacent* duplicate letters that get simplified: `remove_dups("decided")` would return `"decided"`, without any changes, despite the fact that it contains three `d`'s and two `e`'s. Your function should, however, work with any number of adjacent repeated letters. So `remove_dups("brrr")` should return `"br"`.

Solution:

```python
def remove_dups(word):
    new = ""
    for i in range(len(word)):
        if not (i > 0 and word[i] == word[i-1]):
            new += word[i]
    return new
```