CECS 228: Coding Assignment #3

Submission Instructions:

Attach your coded solution to the programming tasks below. When you are finished...

1. Rename this file so that your actual name replaces "YOUR NAME" in the current notebook name, and submit it to the dropbox by **Sunday 11/22 @ 11:59 PM**. For example, I would submit to the dropbox a file called `CECS 228 Coded Assignment #3 - KATHERINE VARELA.ipynb`
2. Submit **your code only to CodePost as `hw3.py` by Sunday 11/22 @ 11:59 PM**

Problem 1:

Use **bitstrings** to complete the implementation of the following functions:

In [6]:
```
def complement(A, U):
    """
    returns the complement of set A, given the universal set U
    raises ValueError if A is not a subset of U
    INPUT:
            - A : subset of elements in U
            - U : universal set of elements
    OUTPUT:
            the set of elements that form the complement of A
    """
    ltA = list(A)
    ltU = list(U)
    bitA = []
    c = set()
    if A.issubset(U):
        for item in range(len(ltU) - 1):
            if ltU[item] in ltA:
                bitA.append(0)
            else:
                bitA.append(1)
        for bit in range(len(bitA) - 1):
            if bitA[bit] == 1:
                c.add(ltU[bit])
        return c
    else:
        raise ValueError(f'{A} is not a subset of {U}')
```

In [7]:
```
def intersection(A, B):
    """
    returns the intersection of sets A and B
    INPUT:
            - A : set of elements
            - B : set of elements
    OUTPUT:
            the set of elements that form the intersection of A and B
    """
    ltA = list(A)
    ltB = list(B)
    iLt = []
    c = set()
    size = 0
    if len(A) > len(B):
        for item in ltA:
            if item in ltB:
                iLt.append(1)
            else:
                iLt.append(0)
        for bit in range(len(iLt) - 1):
            if iLt[bit] == 1:
                c.add(ltA[bit])
        return c
    else:
        for item in ltB:
            if item in ltA:
                iLt.append(1)
            else:
                iLt.append(0)
        for bit in range(len(iLt) - 1):
            if iLt[bit] == 1:
                c.add(ltB[bit])
        return c
```

In [8]:
```
def union(A, B):
    """
    returns the union of sets A and B
    INPUT:
            - A : set of elements
            - B : set of elements
    OUTPUT:
            the set of elements that form the union of A and B
    """
    ltA = list(A)
    bitA = []
    bitB = []
    ltB = list(B)
    u = set()
    if len(ltA) > len(ltB):
        for item in ltA:
            bitA.append(1)
        for item in ltB:
            bitB.append(1)
        for i in range(len(bitA) - 1):
            try:
                if bitA[i] or bitB[i]:
                    u.add(ltA[i])
            except IndexError:
                u.add(ltA[i])
        return u
    else:
        for item in ltA:
            bitA.append(1)
        for item in ltB:
            bitB.append(1)
        for i in range(len(bitB) - 1):
            try:
                if bitA[i] or bitB[i] and ltA[i] < ltB[i]:
                    u.add(ltA[i])
                elif bitA[i] or bitB[i] and ltA[i] > ltB[i]:
                    u.add(ltB[i])
            except IndexError:
                u.add(ltB[i])
        return u
```

In [9]:
```
def difference(A, B):
    """
    returns the difference A-B of sets A and B
    INPUT:
            - A : set of elements
            - B : set of elements
    OUTPUT:
            the set of elements that form the difference of A and B
    """
    ltA = list(A)
    ltB = list(B)
    bitA = []
    d = set()
    for item in range(len(ltA) - 1):
        if item in ltB:
            bitA.append(0)
        else:
            bitA.append(1)
    for bit in range(len(bitA) - 1):
        if bitA[bit] == 1:
            d.add(ltA[bit])
    return d
```

Problem 2:

Complete the function `inverse(f)` which returns the inverse of function f, IF f is a bijection. If f is not a bijection, then the function raises a `ValueError`.

Sample Output

```
>> inverse({(1, 1), (2, 4), (3, 9), (4, 16), (5, 25)})

{(1, 1), (4, 2), (9, 3), (16, 4), (25, 5)}

>> inverse({(1, 1), (2, 4), (3, 9), (4, 16), (5, 25)})

ValueError: Function is not bijective
```

In [6]:
```
def inverse(f):
    """
    returns the inverse of function f
    raises ValueError if function is not bijective
    INPUT:
        - f : a set of tuples representing all pairs (pre-image, image) of the
 function.
    OUTPUT:
        - a set of tuples representing all pairs (image, pre-image) of the inve
rse function.
    """
    set1 = {}
    set2 = {}
    isNone = False

    for i in f:
        j = set1.get(i[0])

        if j is not None and j != i[1]:
            isNone= True
            break
        else:
            set1[i[0]] = i[1]
            a = set2.get(i[1])
        if a is not None and a != i[0]:
            isNone = True
            break
        else:
            set2[i[1]] = i[0]
    if isNone:
        raise ValueError('Function is not bijective')
    inv = [(k, v) for k, v in set2.items()]
    return set(inv)
```

Problem 3:

Complete the function `all_onto_funcs(A, B)` which returns a list of all the onto functions from set A to set B that exist.

Sample Output

```
>> fs = all_onto_funcs({'a', 'b', 'c'}, {1, 2})

>> for i in range(len(fs))

...print(fs[i])

...

>>

{(a, 1), (b, 1), (c, 2)}

{(a, 1), (b, 2), (c, 1)}

{(a, 1), (b, 2), (c, 2)}

{(a, 2), (b, 1), (c, 2)}

{(a, 2), (b, 2), (c, 1)}

{(a, 2), (b, 1), (c, 1)}

>> fs_2 = all_onto_funcs({1, 2}, {'a', 'b', 'c'})

>> print(fs_2)

[]
```

In [1]:
```
import itertools

def all_onto_funcs(A, B):
    """
    returns all onto functions from the set A to the set B
    INPUT:
        - A : a set of elements
        - B : a set of elements
    OUTPUT:
        a list of sets, where each set contains the elements of an onto functio
n from A to B
    """
    all_combos = []
    for element in get_surjective_combinations(B, len(A)):
        all_combos.append(set(zip(A, element)))
    return all_combos

def get_surjective_combinations(choices, n):
    choices = set(choices)
    n_choice = len(choices)
    if n_choice > n:
        return []
    else:
        return [i for i in itertools.product(choices, repeat=n) if set(i) == ch
oices]
```

In [5]:
```
all_onto_funcs({'a', 'b', 'c'}, {1, 2, 3, 4})
```

Out[5]:
```
[]
```

In [ ]:
```

```