Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value
expandtabs()	Sets the tab size of the string
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string
format_map()	Formats specified values in a string
index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
isascii()	Returns True if all characters in the string are ascii characters
isdecimal()	Returns True if all characters in the string are decimals
isdigit()	Returns True if all characters in the string are digits
isidentifier()	Returns True if the string is an identifier
islower()	Returns True if all characters in the string are lower case
isnumeric()	Returns True if all characters in the string are numeric
isprintable()	Returns True if all characters in the string are printable
isspace()	Returns True if all characters in the string are whitespaces
istitle()	Returns True if the string follows the rules of a title
isupper()	Returns True if all characters in the string are upper case
join()	Converts the elements of an iterable into a string
ljust()	Returns a left justified version of the string
lower()	Converts a string into lower case
lstrip()	Returns a left trim version of the string
maketrans()	Returns a translation table to be used in translations
partition()	Returns a tuple where the string is parted into three parts
replace()	Returns a string where a specified value is replaced with a specified value
rfind()	Searches the string for a specified value and returns the last position of where it was found
rindex()	Searches the string for a specified value and returns the last position of where it was found
rjust()	Returns a right justified version of the string
rpartition()	Returns a tuple where the string is parted into three parts
rsplit()	Splits the string at the specified separator, and returns a list
rstrip()	Returns a right trim version of the string
split()	Splits the string at the specified separator, and returns a list
splitlines()	Splits the string at line breaks and returns a list
startswith()	Returns true if the string starts with the specified value
strip()	Returns a trimmed version of the string
swapcase()	Swaps cases, lower case becomes upper case and vice versa
title()	Converts the first character of each word to upper case
translate()	Returns a translated string
upper()	Converts a string into upper case
zfill()	Fills the string with a specified number of 0 values at the beginning

Type	Notation	Example
Addition	+	5 + 6 = 11
Subtraction	-	5 – 6 = -1
Multiplication	*	5 * 6 = 30
Division (float)	/	5 / 6 = 8.3334
Modulus (Remainder after division)	%	For x % y, if x < y, = x 5 % 6 = 5 6 % 5 = 1 Find even numbers: i % 2 == 0
Exponential	**	5 ** 6 = 15625
Floor Division	//	For x // y, if x < y, = 0 5 // 6 = 0 6 // 5 = 1 (always round down to nearest int)

- Three ways to use range() function:
  - range(stop) takes one argument:

```
for i in range(5):
    print(i)
```

  - range(start, stop) takes two arguments:

```
for i in range(3, 10):
    print(i)
```

  - range(start, stop, step) takes three arguments:

```
for i in range(3, 10, 4):
    print(i)
```

```
for i in range(5, 0, -1):
    print(i)
```

```
x = factorial(3)

def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)
```

3*2 = 6 is returned

2*1 = 2 is returned

1 is returned

```
def super_fib_R(term2, n):
    if n == 1:
        return [1]
    elif n == 2:
        return [1, term2]
    else:
        result = super_fib_R(term2, n-1)
        result.append(sum(result))
        return result
```

```
def super_fib_I(term2, upper):
    result = [1, term2]
    total = sum(result)
    while total <= upper:
        result.append(total)
        total *= 2
    return result
```

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

Function	Description
pop()	Remove the item with the specified key.
update()	Add or change dictionary items.
clear()	Remove all the items from the dictionary.
keys()	Returns all the dictionary's keys.
values()	Returns all the dictionary's values.
get()	Returns the value of the specified key.
popitem()	Returns the last inserted key and value as a tuple.
copy()	Returns a copy of the dictionary.

```
{ key : value, key : value }
country_capitals = {
  "United States": "Washington D.C.",
  "Italy": "Rome",
  "England": "London"
}

print(country_capitals["United States"])
 # Washington D.C.

country_capitals["Germany"] = "Berlin
 # add an item with "Germany" as key
and "Berlin" as its value

del country_capitals["United States"]
 # delete item having "United States" key
```

Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Mode	Description
r	Open a file for reading. (default)
w	Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists.
x	Open a file for exclusive creation. If the file already exists, the operation fails.
a	Open a file for appending at the end of the file without truncating it. Creates a new file if it does not exist.
t	Open in text mode. (default)
b	Open in binary mode.
+	Open a file for updating (reading and writing)

- open() - takes two parameters; filename, and mode.
- write() - Writes the specified string to the file
- writelines() - writes a list of strings to the file
- close() - closes the file
- read () - returns the file content
- readable() - returns whether the file stream can be read of not
- readline() - returns one line from the file
- readlines() - returns a list of lines from the file

```
f = open("demofile3.txt", "w")
f.write("CS1010E CHEAT SHEET")
f.close()
```

```
def smallest_second(n):
    """
    n is either term2+1 or a multiple of (term2 + 1)
    Edge case: n = 2. Then we don't divide it anymore!
    Since smallest term2 will be 1 (1+term2 = 2)
    """
    ans = n
    while not n % 2 and n > 2:
        n //= 2
        ans = min(ans, n)

    return ans - 1
```

```
def extract_parentheses_I(stmt):
    result = ""
    for char in stmt:
        if char == '(' or char == ')':
            result += char
    return result
```

```
class PizzaShop:
    def __init__(self, pos, name, radius, hour_s, hour_e):
        self.pos = pos
        self.name = name
        self.radius = radius
        self.hour_s = hour_s
        self.hour_e = hour_e

    def distance_square_to(self, i, j):
        return (self.pos[0] - i) ** 2 + (self.pos[1] - j) ** 2
```

```
def extract_parentheses_R(stmt):
    if len(stmt) == 0:
        return stmt
    else:
        if stmt[0] == '(' or stmt[0] == ')':
            return stmt[0] + extract_parentheses_R(stmt[1:])
        else:
            return extract_parentheses_R(stmt[1:])
```

```
def pd_map(r, c, all_shops, hour):
    map = create_zero_matrix(r, c)
    # Iterate over each cell in the map
    for i in range(r):
        for j in range(c):
            # Initialize the minimum distance to the maximum possible distance
            min_dist = r**2 + c**2
            # Initialize the nearest shop to None
            min_shop = None
            # Iterate over each PizzaShop in all_shops
            for shop in all_shops:
                # Calculate the square of the distance from the shop to the cell
                dist = shop.distance_square_to(i, j)
                # Check if the shop can deliver to the cell at the given hour
                if (dist <= shop.radius**2 and shop.hour_s <= hour < shop.hour_e):
                    # If the distance from the shop to the cell is less than the
                    # current minimum distance
                    if dist < min_dist:
                        # Update the minimum distance and the nearest shop
                        min_dist = dist
                        min_shop = shop
                    # If the distance from the shop to the cell is equal to the
                    # current minimum distance
                    elif dist == min_dist:
                        # Mark the cell as equidistant from multiple shops
                        min_shop = 'X'
            # If the cell is equidistant from multiple shops or not reachable by any shop
            if min_shop == 'X' or min_shop is None:
                # Mark the cell accordingly
                map[i][j] = min_shop if min_shop == 'X' else '.'
            else:
                # If the cell is reachable by a single shop, mark the cell with the
                # first character of the shop's name
                map[i][j] = min_shop.name[0]
    # Return the completed map
    return map
```

```
lst = [1, 2, 3]
# append(): Adds an element at the end of the list.
lst.append(4)  # [1, 2, 3, 4]
# extend(): Adds elements of a list to the end of the current list.
lst.extend([5, 6])  # [1, 2, 3, 4, 5, 6]
# insert(): Adds an element at a specified position.
lst.insert(1, 'a')  # [1, 'a', 2, 3, 4, 5, 6]
# remove(): Removes the first occurrence of the element with the specified value.
lst.remove('a')  # [1, 2, 3, 4, 5, 6]
# pop(): Removes the element at the specified position, or the last item.
lst.pop()  # [1, 2, 3, 4, 5]
# clear(): Removes all the elements from the list.
lst.clear()  # []
# reverse(): Reverses the order of the list.
lst.reverse()  # [5, 4, 3, 2, 1]
len(lst)  # Output: 5
min(lst)  # Output: 1
max(lst)  # Output: 5
print(1 in lst)  # Output: True
print(6 not in lst)  # Output: True

tup = (1, 2, 3, 2, 2, 3)
# count(): Returns the number of times a specified value occurs in a tuple.
tup.count(2)  # Output: 3
# index(): Searches the tuple for a specified value and returns the position.
tup.index(3)  # Output: 2

lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
# slicing [start:stop:step] (stop is exclusive)
# (start defaults to 0) (step defaults to 1)
# if start > stop or start > len(), empty list is returned
lst[1:4]  # Output: [2, 3, 4]
lst[3:]  # Output: [4, 5, 6, 7, 8, 9, 10]
lst[:3]  # Output: [1, 2, 3]
lst[::2]  # Output: [1, 3, 5, 7, 9]
lst[1:5:2]  # Output: [2, 4]
lst[0:6:-1]  # Output: []
lst[::-1]  # Output: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

ASCII

ord('A') -> 65 ord('a') -> 97
chr(65) -> 'A' chr(97) -> 'a'

ord('Z') -> 65 ord('z') -> 122
chr(90) -> 'A' chr(122) -> 'z'

In a dictionary in Python, The values can be any data type, but the key cannot be list, dict, and set.

```
s1 = '1234567890'
print(s1[8:2:-12])  # 9
print(s1[-2:2:-2])  # 975
print(s1[8:-7:-2])  # 975
print(s1[8:3:-2])  # 975
print(s1[-2:3:-2])  # 975
print(s1[8:4:-2])  # 97
```

```
lambda arguments : expression

>>> (lambda x: x + 1)(2)
3
```

CS1010E PE1 Cheat Sheet

Brians Tjipto

found: https://github.com/brianstm/NUS

```python
def equal_to(lhs, rhs, ops):
    """
    1) Get all possible equations (insert ops at different index of lhs)
    2) Eval the expression and check if equal to rhs
    3) For valid expressions, add to set expression + '=rhs'
    """
    ans = set()
    # Convert to list so we can iterate over empty string for convenience
    ops = list(ops) + ['']
    eqns = [lhs[0]]

    # Iteratively get all possible combinations
    for digit in lhs[1:]:
        next_eqns = []
        for eqn in eqns:
            for op in ops:
                next_eqns.append(eqn + op + digit)
        eqns = next_eqns

    # Add to set the valid equations
    for eqn in eqns:
        if eval(eqn) == rhs:
            ans.add(eqn + f'={rhs}')

    return ans
```

```python
def vickrey(bid_file, title, item_num):
    file = read_csv(os.path.join(directory, bid_file))[1:]
    bids = []
    for data in file:
        if data[0] == title and int(data[1]) == item_num:
            bids.append((data[2], float(data[3])))
    if len(bids) == 0:
        return ()
    bids = sorted(bids, key=lambda bid: bid[1], reverse=True)
    return (bids[0][0], bids[1][1]) if len(bids) > 1 else bids[0]
```

```python
def success(bid_file):
    rows = read_csv(os.path.join(directory, bid_file))[1:]
    wins = {}
    for row in rows:
        title, item_num, bidder, price = row[0], int(
            row[1]), row[2], float(row[3])
        winner, winning_price = vickrey(bid_file, title, item_num)
        if bidder == winner:
            if title not in wins:
                wins[title] = {}
            if bidder not in wins[title]:
                wins[title][bidder] = 0
            wins[title][bidder] += 1
    successful_companies = {}
    for title, companies in wins.items():
        max_wins = max(companies.values())
        successful_companies[title] = {
            company for company, wins in companies.items() if wins == max_wins}
    return successful_companies
```

```python
class Player():
    def __init__(self, color):
        self.color = color
        self.wood = 3
        self.brick = 3

    def get_colour(self):
        return self.color

    def get_resources(self):
        return (self.wood, self.brick)

    def robbed(self):
        if self.wood + self.brick < 8:
            return self.color + " is safe from robber"
        else:
            removed_wood = self.wood - (self.wood // 2)
            removed_brick = self.brick - (self.brick // 2)
            self.wood = self.wood // 2
            self.brick = self.brick // 2
            return self.color + " is robbed of " + str(removed_wood + removed_brick) + " resources"

    def trade(self, other_player, woods, bricks):
        if self.wood < woods:
            return self.color + " does not have enough wood"
        elif other_player.brick < bricks:
            return other_player.color + " does not have enough brick"
        else:
            self.wood -= woods
            self.brick += bricks
            other_player.wood += woods
            other_player.brick -= bricks
            return self.color + " trades with " + other_player.color
    def build(self, tile):
        if self.wood < 2 or self.brick < 2:
            return self.color + " does not have enough resources to build"
        elif tile.building > 3:
            return "Tile " + str(tile.number) + " does not have any space"
        else:
            self.wood -= 2
            self.brick -= 2
            tile.add_building(self)
            return self.color + " builds on Tile " + str(tile.number)
```

```python
class Tile():
    def __init__(self, number, resource):
        self.number = number
        self.resource = resource
        self.building = 0
        self.builders = []

    def get_number(self):
        return int(self.number)

    def get_resources(self):
        return self.resource

    def get_building(self):
        return int(self.building)

    def produce(self):
        if self.building == 0:
            return "Tile " + str(self.number) + " does not produce any " + self.resource
        else:
            for player in self.builders:
                if self.resource == "wood":
                    player.wood += 1
                else:
                    player.brick += 1
            return "Tile " + str(self.number) + " produces " + \
                str(self.building) + " " + self.resource

    def add_building(self, player):
        self.building += 1
        self.builders.append(player)
```

```python
def check_balanced(stmt):
    balance = 0
    for char in stmt:
        if char == '(':
            balance += 1
        elif char == ')':
            balance -= 1
        # If at any point balance becomes negative
        # (meaning there's a close parenthesis without
        # a matching open parenthesis)
        if balance < 0:
            return False
    # whether balance is 0, which would indicate that
    # all parentheses are properly balanced
    return balance == 0
```

```python
def most_expensive_burger(money, price_dict):
    change, burger = money, ""
    price_list = sorted(price_dict.items(), key=lambda x: x[1], reverse=True)
    for i in range(len(price_list)):
        if price_list[i][1] > change:
            continue
        else:
            burger += price_list[i][0]
            change = change - price_list[i][1]
    if len(burger) == 0:
        return ("", change)
    else:
        return ("B" + burger + "B", change)
```

```python
def derivative_I(poly):
    result = []
    for i in range(1, len(poly)):
        result.append(i*poly[i])
    return result
```

```python
def derivative_I_2(poly):
    return [poly[1]] + [i * poly[i] for i in range(2, len(poly))]
```

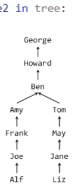
```python
def derivative_R(poly):
    if len(poly) == 1:
        return []
    else:
        return derivative_R(poly[:-1]) + [(len(poly) - 1) * poly[-1]]
```

```python
def compute(poly, x):  # You don't have to understand this, it works
    return sum(poly[i]*x**i for i in range(len(poly)))
```

```python
def newton(poly, x0):
    xn = x0
    while True:
        xn = xn - compute(poly, xn) / compute(derivative_I(poly), xn)
        if abs(compute(poly, xn)) < 0.00001:
            return xn
```

```python
def is_cousin(name1, name2, tree):
    if name1 not in tree or name2 not in tree:
        return False
    if name1 == name2:
        return False
    if tree[name1] == tree[name2]:
        return False
    new_name1, new_name2 = tree[name1], tree[name2]
    if not new_name1 in tree or not new_name2 in tree:
        return False
    return tree[new_name1] == tree[new_name2]
```

```python
def nth_cousin(name1, name2, tree):
    if name1 not in tree or name2 not in tree:
        return -1
    if name1 == name2:
        return -1
    new_name1, new_name2, i = name1, name2, 0
    while True:
        if new_name1 in tree and new_name2 in tree:
            new_name1 = tree[new_name1]
            new_name2 = tree[new_name2]

            if new_name1 == new_name2:
                return i

            i += 1
        else:
            return -1
```




```python
def is_solvable(maze):
    # DFS, but each neighbour is the first reachable stone in 4 directions
    rows, cols = get_row(maze), get_col(maze)

    def get_neighbours(i, j):
        # Given coordinate (i, j), returns a list of at most 4 coordinates
        # Each coordinate is a reachable neighbour (stone) from current coordinate
        neighbours = []
        # Up
        for ii in range(i+1, rows):
            if maze[ii][j]:
                neighbours.append((ii, j))
                break
        # Down
        for ii in range(i-1, 0, -1):
            if maze[ii][j]:
                neighbours.append((ii, j))
        # Right
        for jj in range(j+1, cols):
            if maze[i][jj]:
                neighbours.append((i, jj))
                break
        # Left
        for jj in range(j-1, 0, -1):
            if maze[i][jj]:
                neighbours.append((i, jj))
                break
        return neighbours
    # DFS
    stack = [(0, 0)]
    goal = (rows-1, cols-1)
    visited = set([(0, 0)])
    while stack:
        i, j = stack.pop()
        for neighbour in get_neighbours(i, j):
            if neighbour == goal:
                return True
            if neighbour in visited:
                continue
            stack.append(neighbour)
            visited.add(neighbour)

    return False
```

```python
def step(board):
    rows = len(board)
    cols = len(board[0])
    new_board = create_zero_matrix(rows, cols)
    neighbor = 0
    for i in range(rows):
        for j in range(cols):
            neighbor = 0
            for x in range(max(0, i - 1), min(rows, i + 2)):
                for y in range(max(0, j - 1), min(cols, j + 2)):
                    neighbor += board[x][y]

            neighbor -= board[i][j]

            if board[i][j] == 1 and neighbor <= 1 and neighbor >= 4:
                new_board[i][j] = 0
            if board[i][j] == 1 and (neighbor == 2 or neighbor == 3):
                new_board[i][j] = 1

            elif board[i][j] == 0 and neighbor == 3:
                new_board[i][j] = 1
    return new_board
```

```python
def add_cells(board, cell_list):
    for i in cell_list:
        if 0 <= i[0] < len(board) and 0 <= i[1] < len(board[0]):
            board[i[0]][i[1]] = 1
```

```python
def isMartian(d):
    for i in range(1, d + 1):
        if child_DNA(i) == d:
            return True
    return False
```

```python
def child_DNA(n):
    dna = 1
    for i in str(n):
        dna *= int(i)
    return dna
```

```python
def parent_mutated_DNA(parent_original_dna):
    child_dna = child_DNA(parent_original_dna)
    i = 1
    while True:
        if child_DNA(i) == child_dna:
            return i
        i += 1
```

```python
def prince_map(filename, prince):
    map_data = readmap(filename)
    # Find the prince's position
    for r, row in enumerate(map_data):
        for c, cell in enumerate(row):
            if cell == prince:
                start = (r, c)
                break
    # If the prince's character was not found in the map, return []
    if not start:
        return []
    # Use BFS to find all the connected land cells and their bounds
    minr = minc = float('inf')
    maxr = maxc = float('-inf')
    queue = [start]
    visited = set()
    while queue:
        r, c = queue.pop(0)
        if (r, c) not in visited and 0 <= r < len(map_data) and \
                0 <= c < len(map_data[0]) and map_data[r][c] != 'W':
            minr = min(minr, r)
            maxr = max(maxr, r)
            minc = min(minc, c)
            maxc = max(maxc, c)
            visited.add((r, c))
            queue.extend([(r-1, c), (r+1, c), (r, c-1), (r, c+1)])
    # Crop the map according to the found indices and return the result
    cropped_map = [row[minc:maxc+1] for row in map_data[minr:maxr+1]]
    cropped_map_str = [''.join(row) for row in cropped_map]
    return cropped_map_str
```

```python
def pd_map(r, c, sites):
    map = create_zero_matrix(r, c)
    for i in range(r):
        for j in range(c):
            min_dist = r**2 + c**2
            min_site = -1
            for site_num, site in enumerate(sites):
                dist = (site[0]-i)**2 + (site[1]-j)**2
                if dist < min_dist:
                    min_dist = dist
                    min_site = site_num
                elif dist == min_dist:
                    min_site = 'X'
            map[i][j] = min_site
    return map
```

```python
def crop_map(filename, minr, maxr, minc, maxc):
    mapped = readmap(filename)
    cropped = []
    for i in range(maxr-minr):
        cropped.append(''.join(mapped[minr+i][minc:maxc]))
    return cropped
```

```python
def island_area(filename, prince):
    map_data = readmap(filename)
    # Find the prince's position
    for r, row in enumerate(map_data):
        for c, cell in enumerate(row):
            if cell == prince:
                start = (r, c)
                break
    # If the prince's character was not found in the map, return 0
    if not start:
        return 0
    # Use BFS to find all the connected land cells
    area = 0
    queue = [start]
    visited = set()
    while queue:
        r, c = queue.pop(0)
        if (r, c) not in visited and 0 <= r < len(map_data) and \
                0 <= c < len(map_data[0]) and map_data[r][c] != 'W':
            area += 1
            visited.add((r, c))
            queue.extend([(r-1, c), (r+1, c), (r, c-1), (r, c+1)])
    return area
```

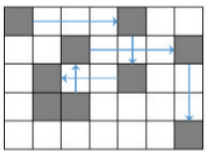
```python
def is_related(name1, name2, pdict):
    family_set = [name1]
    queue = [name1]

    while queue:
        current_member = queue.pop(0)
        for person, parent in pdict.items():
            if person == current_member and parent not in family_set:
                family_set.append(parent)
                queue.append(parent)
            elif parent == current_member and person not in family_set:
                family_set.append(person)
                queue.append(person)

    if name2 in family_set:
        return True

    return False
```

```python
class Necromancer(Mage):
    def __init__(self):
        super().__init__()
        self.name = 'Necromancer'
        self.cost = 400

    def cast(self, my_team, enemy):
        dead_members = [member for member in my_team if not member.alive]
        if dead_members and self.mana >= MANA_COST:
            self.mana -= MANA_COST
            revived_member = random.choice(dead_members)
            revived_member.alive = True
            revived_member.hp = revived_member.maxhp // 2
            dprint(
                f"Reviving member (revived_member.name) with (revived_member.hp) hp")
        else:
            super().cast(my_team, enemy)
```

```python
def encode_I(s1, s2, m):
    result = ''
    for i in range(len(s1)):
        result += chr((((ord(s1[i]) - ord('a')) +
                        int((s2[i])) * m) % 26) + ord('a'))
    return result
```

```python
def encode_R(s1, s2, m):
    if s1 == "":
        return ""
    else:
        return chr((((ord(s1[0]) - ord('a')) + int((s2[0])) * m) %
                    26) + ord('a')) + encode_R(s1[1:], s2[1:], m)
```

```python
def encode_U(s1, s2, m):
    return ''.join(chr((ord(s1[i]) - ord('a') + int(s2[i]) * m)
                       % 26 + ord('a')) for i in range(len(s1)))
```



```python
def per_cipher_i(s, n):
    result = ''
    for i in range(0, len(s), n):
        result += s[i:i+n][::-1]
    return result
```

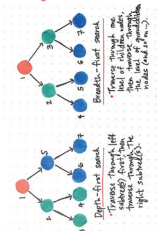
```python
def per_cipher_r(s, n):
    if s == '':
        return s
    else:
        return s[:n][::-1] + per_cipher_r(s[n:], n)
```

```python
def per_cipher_o(s, n):
    return ''.join(s[i:i+n][::-1] for i in range(0, len(s), n))
```

```python
class ArchMage(Mage):
    def __init__(self):
        super().__init__()
        self.name = 'ArchMage'
        self.cost = 600

    def cast(self, my_team, enemy):
        if len([member for member in my_team if member.alive]) == 1:
            dprint('Cast KABOOM to every enemy!')
            for member in enemy:
                if member.alive:
                    member.got_hurt(self.int * 2)
        else:
            super().cast(my_team, enemy)
```

```python
def sum_of_3(L, n):
    # Sort the list
    L = sorted(L)

    # Iterate over the list
    for i in range(len(L) - 2):
        left = i + 1
        right = len(L) - 1

        # Use two pointers to find a pair that sums up to n - L[i]
        while left < right:
            current_sum = L[i] + L[left] + L[right]

            # If the current sum is equal to n, return True
            if current_sum == n:
                return True
            # If the current sum is less than n, move the left pointer to the right
            elif current_sum < n:
                left += 1
            # If the current sum is greater than n, move the right pointer to the left
            else:
                right -= 1

    # If no triplet is found that sums up to n, return False
    return False
```

```python
def merge_sort(lst):
    if len(lst) < 2:  # Base case!
        return lst
    mid = len(lst) // 2
    left = merge_sort(lst[:mid])  #sort left
    right = merge_sort(lst[mid:]) #sort right
    return merge(left, right)
```




```python
from collections import deque

def bfs(graph, start):
    visited = set()
    queue = deque([start])
    while queue:
        vertex = queue.popleft()
        if vertex not in visited:
            visited.add(vertex)
            print(vertex)
            queue.extend(graph[vertex] - visited)
```

```python
visited = set()
def dfs(visited, graph, node):
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)
```

```python
def min_no_of_turns(L):
    if not L:  # If the tuple is empty, no turn is needed
        return 0
    # Convert the tuple to a list and create a dictionary to count each integer
    counts = {}
    for num in L:
        if num in counts:
            counts[num] += 1
        else:
            counts[num] = 1
    turns = 0  # Initialize the count of turns
    # Iterate through the sorted keys of the dictionary
    for num in sorted(counts.keys()):
        # While the count of the current number is not 0
        while counts[num] > 0:
            # Reduce the count of each consecutive number by 1 in each turn
            i = num
            while i in counts and counts[i] > 0:
                counts[i] -= 1
                i += 1
            turns += 1  # Increment the count of turns

    return turns
```

```python
def buyable_map(filename):
    with open(filename, 'r') as f:
        map = [list(line.strip()) for line in f]

    buyable = [['.' for _ in row] for row in map]

    for i in range(len(map)):
        for j in range(len(map[i])):
            if map[i][j] == 'X':
                buyable[i][j] = 'X'
                continue

            for n in [-1, 0, 1]:
                for m in [-1, 0, 1]:
                    if (n != 0 or m != 0) and (0 <= i+n < len(map)) and (0 <= j+m < len(map[i])):
                        if map[i+n][j+m] != map[i][j]:
                            buyable[i][j] = map[i][j]
                            break

    return '\n'.join(''.join(row) for row in buyable)
```

```python
def selection_sort(input_list):
    sort = []
    while input_list:
        smallest = input_list[0]
        for element in input_list:
            if element < smallest:
                smallest = element
        input_list.remove(smallest)
        sort.append(smallest)
    return sort
```