```
# Assignment 1.1

import math

def get_area(radius : float) -> str:
    return radius ** 2 * math.pi

# Ask User to input radius
radius_val = float(input("Enter the radius: "))

# Get the area by running function with user input and display
print("The area is: ", get_area(radius_val))


# Assignment 1.2

# Get strings input by the user
str1 = str(input("Enter first string: "))
str2 = str(input("Enter second string: "))
str3 = str(input("Enter third string: "))

userList = [str1, str2, str3] # Create a list of the strings

def get_last(mylist : list[str]) -> str:
     return mylist[len(mylist)-1]  # Using the length of the list return the last
value accounting for 0 index

print (get_last(userList)) # Print result using function


# Assignment 1.3

value = int(input("Enter the value: ")) # Prompt user for value

def mult1(n : int) -> int:
    return n + n * 11 + n * 111 # Perform calculation

print(mult1(value)) # Call function using user input


# Assignment 1.4

val1 = int(input("Enter the value: ")) # Prompt user for values
val2 = int(input("Enter the value: "))
val3 = int(input("Enter the value: "))

def sum1(n1:int, n2:int, n3:int) -> int:
    if n1 == n2 == n3:
        return (n1 + n2 + n3) * 3 # Return 3 times the number's sum if they are
equal
```

```
    else:
        return n1 + n2 + n3 # Return sum if they are anything but equal

print(sum1(val1, val2, val3))


# Assignment 1.5

userVal = int(input("What is the value: ")) # Prompt user for value

def even_odd(n1:int) -> int:
    if userVal % 2 > 0: # Determine if value is even or odd using modulo operator,
return appropriate value
        return 1
    else:
        return 0
    # return outVal

print (even_odd(userVal)) # Call function using the user's input value


# Assignment 1.6

userVal = str(input("Enter the letter: "))  # User is prompted to enter a letter

def is_vowel(n1:chr) -> bool:
    vList = ['a', 'e', 'i', 'o', 'u']
    for x in vList: # Loop through each vowel to determine if the user letter
matches
        if x == n1:
            return True
    return False

print (is_vowel(userVal))   # Print statment calling the function with user input


# Assignment 1.7

from typing import Any

userList = ['foo', 'bar', False, 99.3, True]

def make_strl(mylist:list) -> str:
    strOut = "" # Create an empty String variable to contain the final string
    for i in mylist:    # Check if each item in the user's list is a string
        if type(i) == str:
            strTemp = i # If True store the value in a temporary variable
        else:
            i = str(i)  # If False convert the value to string and add to temporary
variable
```

```
            strTemp = i
        strOut = strOut + strTemp   # Create the final string by adding the 
temporary variable value
    return strOut

print(make_strl(userList))  # Use print to call the function with user values


# Assignment 1.8

userList1 = ["White", "Black", "Red"]
userList2 = ["Red", "Green"]

# Syntax as written in the instructions "color_list_1 : List[str]" doesn't seem to 
work, but simply "list" is ok
def extract1(color_list_1 : list, color_list_2 : list) -> set:
    mySet = set()   # Create an empty set variable to hold the final result
    for i in color_list_1:  # Check each item in userList1 against userList2
        if i not in color_list_2 and i not in mySet:
            mySet.add(i)    # If item from color_list_1 is not in color_list_2 and 
not previously added to mySet, add it
    return mySet

print(extract1(userList1, userList2))   # Print function result with user values


# Assignment 1.9

# Get number inputs from the user
numb1 = int(input("Enter the first number: "))
numb2 = int(input("Enter the second number: "))

def addem (n1: int, n2: int) -> bool:
    if n1 == n2 or abs(n1 - n2) == 5 or abs(n1 + n2) == 5: # Perform comparision
        return True
    else:
        return False

print(addem(numb1, numb2)) # Print statement calling the function


# Assignment 1.10

def solve1(x : int, y : int) -> int: # Define function
    return (x + y) ** 2

# Prompt user for input
xIntercept = int(input("Please enter the 'x' intercept: "))
yIntercept = int(input("Please enter the 'y' intercept: "))
```

```
print(solve1(xIntercept, yIntercept)) # Call function in print statement w user
input


# Assignment 1.11

def solve1(x : int, y : int) -> int:
    return (x + y) ** (x + y)

# Prompt the user for inputs
xInt = int(input("Please enter the x intercept: "))
yInt = int(input("Please enter the y intercept: "))

print(solve1(xInt, yInt)) # Call function in print statement with user inputs


# Assignment 1.12

import math
def solve1(p1 : tuple, p2 : tuple) -> int:
    return math.sqrt((p2[0]-p1[0]) ** 2 + (p2[1]-p1[1]) ** 2)

# Prompt user for input
x1 = int(input("Enter x intercept 1: "))
y1 = int(input("Enter y intercept 1: "))
x2 = int(input("Enter x intercept 2: "))
y2 = int(input("Enter y intercept 2: "))

# Place user inputs into variables type tuple to match what function is expecting
point1 = x1, y1
point2 = x2, y2

print(solve1(point1, point2))


# Assignment 1.13

# Variable for test cases
ithList = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6]

def is_ith(p1 : list) -> bool:
    bList = []  # Create an empty list that will hold bool values
    for i in ithList: # Loop through each value in the test case
        if ithList.count(i) == i: # Count quantity of occurrences of value and check
if it matches value
            bList.append(True)
        else:
            bList.append(False)
    return all(bList) #Use all() function to verify if every value in bList is true
to return correct result
```

```
print(is_ith(ithList)) # Call function in print statement


# Assignment 1.14

def longest_str (p1: list) -> str:
    longestWordLength = 0 # Create 0 value variable to compare against which ensures
first word will orver-write
    for i in p1: # Loop through each value provided in test cases
        if len(i) > longestWordLength: # if value is longer than check variable,
replace with this value
            longestWord = i
            longestWordLength = len(i)
    return longestWord

wordList = ['cat', 'dog', 'shatter', 'donut', 'at', 'todo', ''] # Provided test case

print(longest_str(wordList)) # Call function in print statement
```

```
# Assignment 1.14

def longest_str (p1: list) -> str:
    longestWordLength = 0 # Create 0 value variable to compare against which ensures
first word will orver-write
    for i in p1: # Loop through each value provided in test cases
        if len(i) > longestWordLength: # if value is longer than check variable,
replace with this value
            longestWord = i
            longestWordLength = len(i)
    return longestWord

wordList = ['cat', 'dog', 'shatter', 'donut', 'at', 'todo', ''] # Provided test case

print(longest_str(wordList)) # Call function in print statement
```

```
# Assignment 1.15

def split_str (s:str) -> list:
    userStrList = [] # Create an empty variable list to hold the result
    if ' ' not in s and ',' not in s: # The first case checks if spaces and commas
are not in the string
        for i in s:
            userStrList.append(i) # In this case for each letter in the string add
it individually to the list
        userStrList = userStrList[::-1] # Reverse the order of the list
    elif ',' in s: # Handles splitting the string if there is a comma
        userStrList = [s.split(',')]
    else: # Final case handles splitting the string if there is a space
        userStrList = [s.split(' ')]

    return userStrList

userString = str(input("Enter your string: ")) # Prompt user for string input
print(split_str(userString))


# Assignment_1.16

userList = list(input("Enter your list of numbers: ")) # Prompt user for list of
numbers
userNumber = int(input("Enter the number of high numbers: ")) # Prompt user for
quantity of high numbers to return

def largest_k(mylist: list, knum: int) -> list:
    for i in mylist: # Checks each item in the list to verify they are digits
        if i.isdigit() == False:
            mylist.remove(i) # Remove items that if they are not digits

    mylist = [int(i) for i in mylist] # Forces each item in the list to be an
integer

    highList = [] # Establish an empty list variable to return the result
    c = 0 # Counter variable
    while c < knum: # While loop to ensures user designated quantity of values is
returned
        maxValue = 0 # Establish a max value for comparison as 0 to ensure it'll be
overwritten for positive numbers
        for i in mylist:
            if maxValue < i: # Compares max value to current value
                maxValue = i  # Reassigns max value to current value if current
value is higher
        c += 1 # Counter advance
        highList.append(maxValue) # Adds the values to list to return to user
        mylist.remove(maxValue) # Ensures same result isn't returned more than once

    return highList

print(largest_k(userList, userNumber))


# Assignment 1.17

userList = [1, 2, 3, 4, 5, 1]

def all_different (p1: list) -> bool:
# Function forces user values from a list to a set.
# Sets don't allow duplicates so if the length is different some numbers were
```

```
inherently the same
    if len(p1) != len(set(p1)):
        return False
    else:
        return True

print (all_different(userList))


# Assignment 1.18:

userList = [1,2,3,4,4,5,5,6]

# print (userList)

def remove_dups (p1: list) -> list[int]:
    newSet = set(p1)

    return list(newSet)

print(remove_dups(userList))


# Assignment 1.19

gradeList = [5.0, 4.7, 3.4, 3.0, 2.7, 2.4, 2.0, 1.7, 1.4, 0.0]

def grades_to_letter (p1: list) -> list:
    letterGradeList = [] # Variable to hold final return values
    for x in p1: # Loop through each GPA in the list
        # Use if statements to assign letter grades and append those to the grade
list
        if x >= 4.0:
            letterGradeList.append('A+')
        elif x >= 3.7 and x < 4.0:
            letterGradeList.append('A')
        elif x >= 3.4 and x < 3.7:
            letterGradeList.append('A-')
        elif x >= 3.0 and x < 3.4:
            letterGradeList.append('B+')
        elif x >= 2.7 and x < 3.0:
            letterGradeList.append('B')
        elif x >= 2.4 and x < 2.7:
            letterGradeList.append('B-')
        elif x >= 2.0 and x < 2.4:
            letterGradeList.append('C+')
        elif x >= 1.7 and x < 2.0:
            letterGradeList.append('C')
        elif x >= 1.4 and x < 1.7:
            letterGradeList.append('C-')
        else:
            letterGradeList.append('F')
    return letterGradeList

print(grades_to_letter(gradeList)) # Call function in print statement using values
defined above


# Assignment 1.20

wordList = ['ably', 'abruptly', 'abecedary', 'apparently', 'acknowledgedly']

def vowelFinder (p1: list) -> list:
    vResult = [] # Empty list to hold returned values
    vString = '' # Variable to hold temporary values to append to the list
    counter = 0

    for i in p1: # Loop through each item in the list 'P1'
        while counter < len(i): # Use counter value to step through each letter of
each item
            # Use if statements to add vowels to temporary variables
            if i[counter] == 'a':
                vString = vString + 'a'
            elif i[counter] == 'e':
                vString = vString + 'e'
            elif i[counter] == 'i':
                vString = vString + 'i'
            elif i[counter] == 'o':
                vString = vString + 'o'
            elif i[counter] == 'u':
                vString = vString + 'u'
            elif i[counter] == 'y' and len(i)-1 == counter:
                vString = vString + 'y'
            counter += 1
        counter = 0 # Reset the letter counter for the next list item
        vResult.append(vString) # Add the new strings of vowels to the returned list
        vString = '' # Reset the temporary string variable

    return vResult

print(vowelFinder(wordList)) # Call function in print statement
```