```
#Converting a String to a List, use the join() method in str class to concatenate all the characters.
s1="WORD"
print ("original string:", s1)
l1=list(s1)
l1.insert(3,"L")
print (l1)
s1=''.join(l1)
print ("Modified string:", s1)
```

```
original string: WORD
['W', 'O', 'R', 'L', 'D']
Modified string: WORLD
```

```
#String Concatenation
str1="Hello"
str2="World"
print ("String 1:",str1)
print ("String 2:",str2)
str3=str1+str2
print("String 3:",str3)
```

```
String 1: Hello
String 2: World
String 3: HelloWorld
```

```
#To insert a whitespace between the two, use a third empty string.
str1="Hello"
str2="World"
blank=" "
print ("String 1:",str1)
print ("String 2:",str2)
str3=str1+blank+str2
print("String 3:",str3)
```

```
String 1: Hello
String 2: World
String 3: Hello World
```

```
#Python program to find number of vowels in a given string.
mystr = "All are Human Beings and are equal"
vowels = "aeiou"
count=0
for x in mystr:
   if x.lower() in vowels:
    count+=1
print ("Number of Vowels:", count)
```

```
Number of Vowels: 13
```

```
#Python program to convert a string with binary digits to integer.
mystr = '1101'
def binint(mystr):
   for x in mystr:
      if x not in '01':
        return "Error. String with non-binary characters"
   num = int(mystr, 2)
   return num
print ("binary:{} integer: {}".format(mystr,binint(mystr)))
```

```
binary:1101 integer: 13
```

```
#Python program to drop all digits from a string.
digits = [str(x) for x in range(10)]
mystr = 'He12llo, Py00t234h55on!'
chars = []
for x in mystr:
   if x not in digits:
      chars.append(x)
newstr = ''.join(chars)
print (newstr)
```

```
Hello, Python!
```

```
def convert_distance(miles):
    km = miles * 1.6  # approximately 1.6 km in 1 mile
    return km

# Do not indent any of the following lines of code as they are
# meant to be located outside of the function above

my_trip_miles = 55

# 2) Convert my_trip_miles to kilometers by calling the function above
my_trip_km = convert_distance(my_trip_miles)

# 3) Fill in the blank to print the result of the my_trip_km conversion
print("The distance in kilometers is " + str(my_trip_km))
```

```
The distance in kilometers is 88.0
```

```
number = 4
if number * 4 < 15:
 print(number / 4)
elif number < 5:
 print(number + 3)
else:
 print(number * 2 % 5)
```

```
7
```

```
n = 4
if n*6 > n**2 or n%2 == 0:
    print("Check")
```

```
Check
```

```
def greater_value(x, y):
    if x > y:
        return x
    else:
       return y


print(greater_value(10,3*5))
```

```
15
```

```
def complementary_color(color):
    if color == "blue":
        complement = "orange"
    elif color == "yellow":
        complement = "purple"
    elif color == "red":
        complement = "green"
    else:
        complement = "unknown"
    return complement

print(complementary_color("blue")) # Should print orange
print(complementary_color("yellow")) # Should print purple
print(complementary_color("red")) # Should print green
print(complementary_color("black")) # Should print unknown
print(complementary_color("Blue")) # Should print unknown
print(complementary_color("")) # Should print unknown
```

```
orange
purple
green
unknown
unknown
unknown
```

```
def greater_value(x, y):
    if x > y:
        return x
    else:
       return y


print(greater_value(10,3*5))
```

```
15
```

```
def sum_divisors(number):
  # Initialize the appropriate variables
  total = 0
  divisor = 1

  # Avoid dividing by 0 and negative numbers
  # in the while loop by exiting the function
  # if "number" is less than one
  if number < 1:
    return 0

  # Complete the while loop
  while divisor>= 1:
    if number % divisor == 0:
      total += divisor
    # Increment the correct variable
    total += 1

  # Return the correct variable
  return total
print(sum_divisors(0)) # Should print 0
print(sum_divisors(3)) # Should print 1
# 1
print(sum_divisors(36)) # Should print 1+2+3+4+6+9+12+18
# 55
print(sum_divisors(102)) # Should print 1+2+3+6+17+34+51
# 114
```

```
    0
```

```
def difference(x, y):
    z = x - y
    return z
print(difference(5, 3))
```

```
    2
```

```
#List Comprehensions
[(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
```

```
    [(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

```
# create a new list with the values doubled
l1 = [-4, -2, 0, 2, 4]

[x*2 for x in l1]
```

```
    [-8, -4, 0, 4, 8]
```

```
# filter the list to exclude negative numbers
l1 = [-4, -2, 0, 2, 4]
[x for x in l1 if x >= 0]
```

```
    [0, 2, 4]
```

```
# apply a function to all the elements
l1 = [-4, -2, 0, 2, 4]
[abs(x) for x in l1]
```

```
    [4, 2, 0, 2, 4]
```

```
# create a list of 2-tuples like (number, square)

[(x, x**2) for x in range(6)]
```

```
    [(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25)]
```

```
# the tuple must be parenthesized, otherwise an error is raised

[x, x**2 for x in range(6)]
```

```
      File "<ipython-input-14-67d70077821e>", line 3
        [x, x**2 for x in range(6)]
         ^
    SyntaxError: did you forget parentheses around the comprehension target?
```

SEARCH STACK OVERFLOW

```
#List comprehensions can contain complex expressions and nested functions:
from math import pi
```

```
[str(round(pi, i)) for i in range(1, 6)]

     ['3.1', '3.14', '3.142', '3.1416', '3.14159']
```

```
#Using Lists as Stacks
stack = [3, 4, 5]
stack.append(6)
stack.append(7)

print(stack)

stack.pop()

     [3, 4, 5, 6, 7]
     7
```

```
#Using Lists as Queues
from collections import deque
queue = deque(["Eric", "John", "Michael"])
queue.append("Terry")           # Terry arrives
queue.append("Graham")          # Graham arrives
print(queue)

queue.popleft()                 # The first to arrive now leaves
queue.popleft()                 # The second to arrive now leaves
print(queue)                           # Remaining queue in order of arrival


     deque(['Eric', 'John', 'Michael', 'Terry', 'Graham'])
     deque(['Michael', 'Terry', 'Graham'])
```

```
#A tuple consists of a number of values separated by commas:
t = 12345, 54321, 'hello!'
t[0]

     12345
```

```
#A tuple consists of a number of values separated by commas:
t = 12345, 54321, 'hello!'
# Tuples may be nested:
u = t, (1, 2, 3, 4, 5)
u

     ((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
```

```
# Tuples are immutable:
t = 12345, 54321, 'hello!'
t[0] = 88888

     ---------------------------------------------------------------------------
     TypeError                                 Traceback (most recent call last)
     <ipython-input-3-962cce3038eb> in <cell line: 3>()
           1 # Tuples are immutable:
           2 t = 12345, 54321, 'hello!'
     ----> 3 t[0] = 88888

     TypeError: 'tuple' object does not support item assignment
```

SEARCH STACK OVERFLOW

```
# but they can contain mutable objects:
v = ([1, 2, 3], [3, 2, 1])

v

     ([1, 2, 3], [3, 2, 1])
```

```
# special problem is the construction of tuples containing 0 or 1 items: the syntax has some extra quirks to accommodate these.
#Empty tuples are constructed by an empty pair of parentheses;
#a tuple with one item is constructed by following a value with a comma
empty = ()
len(empty)

#singleton = 'hello',    # <-- note trailing comma
#len(singleton)
```

```
0
```

```
#A set is an unordered collection with no duplicate elements
basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
print(basket)

#Curly braces or the set() function can be used to create sets. To create an empty set you have to use set(), not {}
```

```
{'orange', 'banana', 'pear', 'apple'}
```

```
# Demonstrate set operations on unique letters from two words

a = set('abracadabra')
print(a)                #unique letters in a
b = set('alacazam')
print(b)                #unique letters in b
print(a - b)                   # letters in a but not in b
print(a | b)                  # letters in a or b or both
print(a & b)                   # letters in both a and b
print(a ^ b)                   # letters in a or b but not both

#Similarly to list comprehensions, set comprehensions are also supported
a = {x for x in 'abdbra' if x not in 'abc'}
print(a)
```

```
{'r', 'b', 'a', 'c', 'd'}
{'a', 'z', 'm', 'c', 'l'}
{'d', 'b', 'r'}
{'b', 'a', 'z', 'c', 'm', 'l', 'r', 'd'}
{'a', 'c'}
{'l', 'b', 'r', 'z', 'd', 'm'}
{'r', 'd'}
```

```
for n in range(10):
    print(n+n)
```

```
0
2
4
6
8
10
12
14
16
18
```

```
def is_power_of_two(number):
  # This while loop checks if the "number" can be divided by two
  # without leaving a remainder. How can you change the while loop to
  # avoid a Python ZeroDivisionError?
  while number % 2 == 0:
    number = number / 2
    break
  # If after dividing by 2 "number" equals 1, then "number" is a power
  # of 2.

  if number == 1:
    return True

  return False


# Calls to the function
print(is_power_of_two(0)) # Should be False
print(is_power_of_two(1)) # Should be True
print(is_power_of_two(8)) # Should be True
print(is_power_of_two(9)) # Should be False
```

```
False
True
False
False
```

+ Code + Text

```
def multiplication_table(start, stop):
    # Complete the outer loop range
    for x in range(1,3):
         # Complete the inner loop range
        for y in range(3,1):
            # Prints the value of "x" multiplied by "y"
            # and inserts a space after each value
            print(str(x*y), end=" ")
        # An empty print() function inserts a line break at the
        # end of the row
        print()


multiplication_table(1, 3)
```

```
animal="Hippopotamus"
print(animal[3:6])
print(animal[-5])
print(animal[10:])
```

```
pop
t
us
```

```
car_makes = ["Ford", "Volkswagen", "Toyota"]
car_makes.remove("Ford")
print(car_makes)
```

```
['Volkswagen', 'Toyota']
```

```
teacher_names = {"Math": "Aniyah Cook", "Science": "Ines Bisset", "Engineering": "Wayne Branon"}
teacher_names.values()
```

```
dict_values(['Aniyah Cook', 'Ines Bisset', 'Wayne Branon'])
```

```
def isComposite(x):
  if x > 1:
    for i in range(2, x):
     if(x % i == 0):
       return True
     else:
       return False
isComposite(9)
```

```
False
```

```
def isPerfect( x ):
    sum = 1
    i = 2
    while i * i <= x:
        if x % i == 0:
            sum = sum + i + x/i
        i += 1

    # If sum of divisors is equal to
    # n, then n is a perfect number

    return (True if sum == x and x!=1 else False)
isPerfect( 28 )
isPerfect( 6 )
```

```
True
```

```
def isComposite(x):
    if x > 1:
        for i in range(2, x):
            if(x % i == 0):
                return True
            else:
                return False
isComposite(9)
isComposite(22)
isComposite(3)
isComposite(41)
```

```
False
```

```
def isAbundantNumber(n):
    # To store the sum of divisors
    sum_divisors = 1
    # Loop through the numbers [2, sqrt(n)]
    i = 2
    while i * i <= n:
        if n % i == 0:
            if i * i != n:
                sum_divisors += i + n // i
            else:
                sum_divisors += i
        i += 1

    if sum_divisors > n:
        return True
    else:
        return False

isAbundantNumber(12)
```

```
True
```

```
def isNarcissistic(x):
    # your code here
    logical = True
    logical2 = True
    i = 0
    j = 0
    notation = 10
    sum = 0

    #Calculating the number notation
    while logical:
        if 10 ** i <= x:
            notation = 10 ** i
            i = i + 1
        else:
            logical = False

    #i from now on is also the qauntity of digits
    while logical2:
        if ( notation / 10 ** j ) >= 1:
            sum = sum + (( x // ( notation / 10 ** j ) ) % 10) ** i
            j = j + 1
        else:
            logical2 = False

    if sum == x:
        return True
    else:
        return False

isNarcissistic(153)
```

```
True
```