List comprehension

In [1]:
```
# create a list of ten number using list comprehension
num = [x for x in range(100)] #generate a list from 0 to 100
print(num)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,
42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61,
62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81,
82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]
```

In [2]:
```
# using if statement within list comprehension
nums = [x for x in range(10) if x % 2 ==0 ] #generate list of even nums
print(nums)
```

```
[0, 2, 4, 6, 8]
```

In [3]:
```
# using if/else statement within list comprehension
nums = ["Even" if x % 2 == 0 else "Odd" for x in range(10)]
print(nums)
```

```
['Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd']
```

In [4]:
```
# creating a list of squared numbers from another list of numbers using list comp
nums = [2,4,6,8]
squared_nums = [nums**2 for nums in nums]
print(squared_nums)
```

```
[4, 16, 36, 64]
```

In [5]:
```
# creating a dictionary of even numbers and square values using comprehension
numbers = [x for x in range(10)]
squares = {num: num**2 for num in numbers if num %2==0}
print(squares)
```

```
{0: 0, 2: 4, 4: 16, 6: 36, 8: 64}
```

In [15]:
```
"""
Exercise: Using list comprehension, convert the following list to
Fahrenheit. Currently, the degrees are in Celsius temperatures. The conversion
formula is “(9/5) * C + 32”. Your output should be [ 53.6, 69.8, 59, 89.6 ].
>>> degrees = [ 12, 21, 15, 32 ]
"""
degrees = [12, 21, 15, 32]

degrees = [round((9/5) * deg + 32, 1) for deg in degrees]

print(degrees)
```

```
[53.6, 69.8, 59.0, 89.6]
```

In [20]:
```
"""
Exercise: Ask the user to input a single integer up to and including 100.
Generate a list of numbers that are exactly divisible by that number up to and
including 100 using list comprehension. For example, if the number 25 was
input, then the output should be [ 25, 50, 75, 100 ].
"""
ask =int( input("Enter a number between 1 to 100(including): "))
listy = [num for num in range(1,101) if num % ask == 0 ]
print(listy)
```

```
Enter a number between 1 to 100(including): 25
[25, 50, 75, 100]
```

Lambda Function

In [21]:
```
# using a lambda function
( lambda x : x**2)(4)
# first parenthesis is lambda function and second is the argument
```

Out[21]:
```
16
```

In [22]:
```
# passing multiple arguments into a lambda
(lambda x,y : x * y)(10,5)
# returns x * y
```

Out[22]:
```
50
```

In [23]:
```
# saving a lamvda function into a variable
square = lambda x,y : x*y
print(square)
result = square(10,5) #calls the lamda function
print(result)
# no need of parenthesis lambda function when using variables
```

```
<function <lambda> at 0x000001D10C126D30>
50
```

In [24]:
```
# using if/else within a lambda to return the greatest number
greater = lambda x,y : x if x > y else y
result = greater(5,10)
print(result)
```

```
10
```

In [26]:
```
# returning a lambda function from another function
def my_func(n):
    return lambda x : x * n
doubler = my_func(2) # returns equivalent of lambda x : x * 2
print(doubler(5)) # will output 10
tripler = my_func(3)
print(tripler(5))
```

```
10
15
```

In [28]:
```
"""
Exercise: Write a lambda function that
takes in a degree value in Celsius and returns the degree converted into Fahrenhe
"""
(lambda deg : (9/5)* deg + 32) (22)
```

Out[28]:
```
71.6
```

Map , Reduce and Filter

In [32]:
```
# using the map function without lambdas
def convDeg(c):
    return round((9/5) * c + 32, 1)
temp = [12.5, 13.6, 9.2]
converted_temp = map(convDeg , temp)
print(converted_temp) # map function returns a map object, not converted data col
converted_temp = list(converted_temp) #type convert map object
print(converted_temp)
```

```
<map object at 0x000001D10D387400>
[54.5, 56.5, 48.6]
```

In [34]:
```
# using a map function with lambdas
temp = [12.5, 13.6, 9.2]
converted_temp = list(map(lambda c :round((9/5) * c + 32, 1 ), temp))
print(converted_temp)
```

```
[54.5, 56.5, 48.6]
```

In [2]:
```
# using the filter function without lambda function, filter  out temps below 55f
def filterTemps(c):
    converted = (9/5) * c +32
    return True if converted > 55 else False  # use ternary operator
temps = [12, 14, 15, 9]
filtered_temp = filter(filterTemps, temps)
print(filtered_temp)
filtered_temp = list(filtered_temp)
print(filtered_temp)
```

```
<filter object at 0x0000017DF5E0C0D0>
[13.6, 15]
```

In [4]:
```
# using lambda function to filter function temperature below 55F
temps = [12, 15, 16, 14, 10, 3 ,17]
filtered_temps = list(filter(lambda c : True if (9/5) * c + 32 > 55 else False, t
print(filtered_temps)
```

```
[15, 16, 14, 17]
```

In [6]:
```
# use the reduce function
from functools import reduce
nums = [1,2,3,4]
result = reduce(lambda a,b : a * b, nums)
print(result)
```

```
24
```

In [12]:
```
"""
Exercise: Use a lambda and map function to map over the list of
names in the following to produce the following result “[ “Ryan”, “Paul”,
“Kevin Connors” ].
>>> names = [ " ryan", "PAUL", "kevin connors " ]
"""
names = [ "ryan", "PAUL", "kevin connors " ]
output = list(map(lambda name : name.title(),names))
print(output)
```

```
['Ryan', 'Paul', 'Kevin Connors ']
```

In [13]:
```
"""
Exercise: Using a lambda and filter function, filter out all the names that
start with the letter “A.” Make it case insensitive, so it filters out the name
whether it’s uppercase or not. The output of the following list should be
[ “Frank”, “Ripal” ].
>>> names = [ "Amanda", "Frank", "abby", "Ripal", "Adam" ]
"""
names = ["Amanda", "Frank", "abby", "Ripal", "Adam"]

names = list(filter(lambda name: True if name[0].lower() != 'a' else False, names

print(names)
```

```
['Frank', 'Ripal']
```

Recursive Function and Memoization

In [7]:
```
# writing a factorial using recursive function
def factorial(n):
    # set your base case!
    if n <= 1:
        return 1
    else:
        return factorial(n-1) * n
print(factorial(5))
```

```
120
```

In [17]:
```
# writing a recursive fibonacci sequence
def fib(n):
    if n <=1:
        return n
    else:
        return fib(n-1)+ fib(n-2)
print(fib(5))
```

```
5
```

In [9]:
```
# using memoization with the fibonacci sequence
cache = {} # using to cache values to be used later
def fib(n):
    if n in cache :
        return cache[n]  #return value stored in dictionary
    result = 0
    # base case
    if n <= 1:
        result = n
    else:
        result = fib(n-1) + fib(n-2)
    cache[n] = result

    return result
print(fib(50)) #calculates almost instantly
```

```
12586269025
```

In [10]:
```
# using @Lru_cache, Python's default moization/caching technique
from functools import lru_cache
@lru_cache()   # python's built-in memoization system
def fib(n):
    if n <= 1:
        return n
    else:
        return fib( n -1 ) + fib(n-2)
fib(50)
```

Out[10]:
```
12586269025
```

Binary Search Algorithm

In [23]:

```python
# setting up imports and generating a list of random number
import random
nums = [random.randint(0,20) for i in range(20)]

def binarySearch(aList, num):
    #step1 : sort the list
    aList.sort()
    #step6: setup a loop to repeat step 2 through 6 until list empty
    while aList:
        #step2 : find the middle index
        mid = len(aList) // 2 # two slashes mean floor division.
        #step3 : check the value at middle index, if equal to num return Tru
        if aList[mid] == num:
            return True
        #step4 : check if value is greater, if so,cut off right half of list
        elif aList[mid] > num:
            aList = aList[ : mid ]

        #step5: check if value is less,if so, cut off left half
        elif aList[mid] < num:
            aList = aList[ mid + 1 : ]
        print(aList)
        #step 7:return False if it makes to this line it means list was empt
    return False

print(sorted(nums))
print(binarySearch(nums,3))
```

```
[0, 1, 3, 4, 4, 5, 9, 9, 9, 10, 10, 11, 12, 12, 12, 16, 16, 17, 18, 20]
[0, 1, 3, 4, 4, 5, 9, 9, 9, 10]
[0, 1, 3, 4, 4]
True
```

In [ ]:
```
# Binary Search using Recursion
import random

nums = [ random.randint(0, 20) for i in range(10) ]

def binarySearch(aList, num):
    # base case
    if aList == []:
        return False

    aList.sort( )

    mid = len(aList) // 2

    if aList[mid] == num:
        return True
    elif aList[mid] > num:
        return binarySearch(aList[ : mid ], num)
    elif aList[mid] < num:
        return binarySearch(aList[ mid + 1 : ], num)

print( sorted(nums) )
print( binarySearch(nums, 3) )
```