Introduction to Python Programming

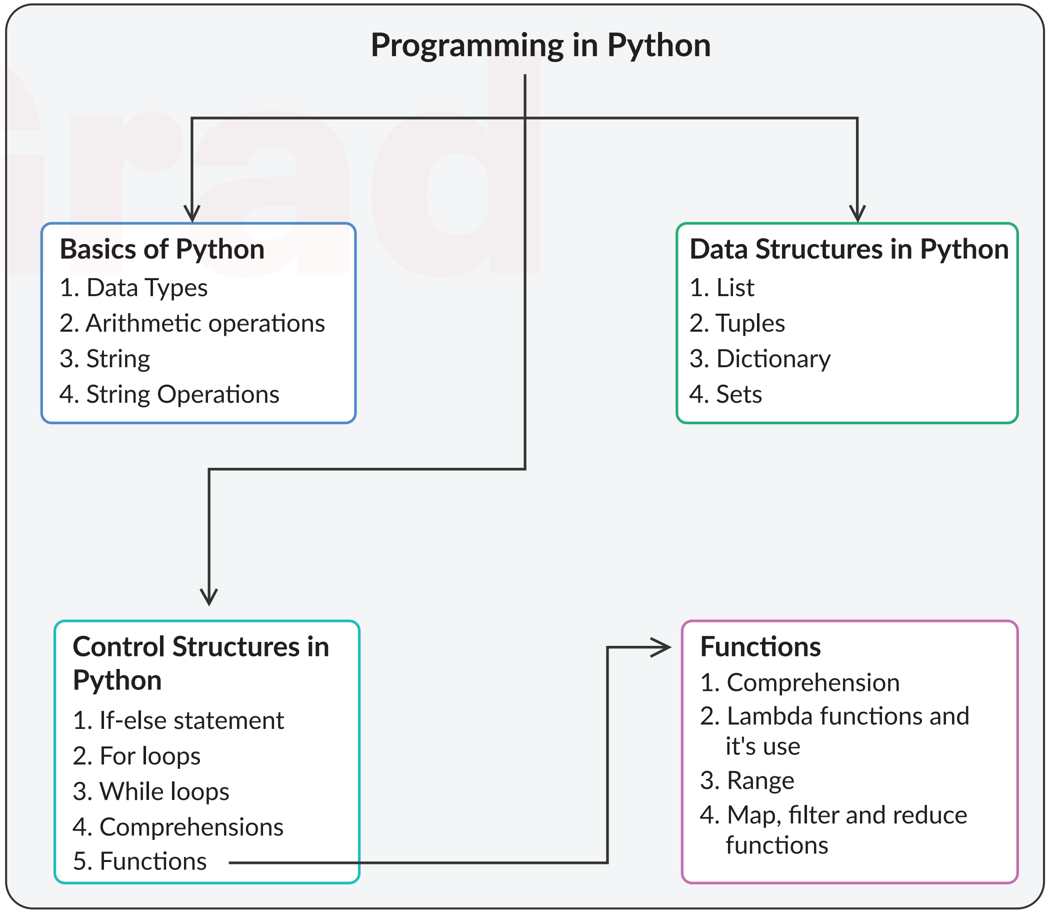
You are expected to handle and analyse large volumes of data to draw inferences, and Python serves as an easy and effective tool to run different operations. It is widely used in the scientific and research communities because of the simple syntax and a rich collection of libraries dedicated to various aspects of data science.

As part of Python programming, you covered:

- Data and its types: integer, float, string, Boolean
- Data structures such as list, tuples etc. used to store data in Python
- Various operations that can be performed over lists, tuples, dictionaries and sets
- Control structures to help in decision making such as if-else statements, for and while loop
- Define functions to execute repetitive tasks in Python

Common Interview Questions:

1. What is the difference between a module and a package in Python?
2. How to convert a number into a string?
3. What is lambda function in Python?
4. Explain the difference between a list and a tuple?
5. How do you change the data type of a list?
6. What is the difference between generators and iterators?
7. What is the range() function and wha t are its parameters?
8. Is Python case-sensitive?
9. What are functions?
10. How do you do data abstraction in Python?
11. What are the escape sequences in Python?
12. What is an Expression?



Introduction to Python Programming

Data Type	Syntax
Integer	int
Float	float
String	str
Boolean	bool

Arithmetic

```
#Float division
133//10     #13

#Modulus division
133%10       #3

#PEMDAS rule
3*3-(4+3)//2 #6
```

String

```
#Syntax: 'string' or "string"
#\ is escape character:
'I don\'t know'
#\n-newline character
print('Line1 \n Line2')
str1 = 'first'
str2 = 'last'
#concatenation
str3 = str1+str2 #firstlast
#Access any character
str1[1] #f
str1.upper() #FIRST
str1.lower() #first
str1.replace('f','th') #thirst
str1.count('f')  #1
```

Data Structures	List	Tuples	Sets
Definitions	Lists are like arrays but it can contains different types of data together whereas arrays can hold same type of data	Tuples are containers for holding different types of values	Contains all types of data with no duplicate entry
Mutable/ Immutable	Mutable	Immutable	Mutable
Example	`list_1 = [1,2,3,4,5]`	`tuple_1 = (1,2,3,4,5)`	`set_1 = {1,2,3,4,5}`

Indexing	`list_1[1] #2nd value - 2`	`list_1[1]`
		`#2nd value - 2`
			Is unordered: hence index of element is not defined on sets

Length	`len(list_1) #5`	`len(list_1) #5`	`len(list_1) #5`
Sum of elements	`sum(list_1) #15`	`sum(tuple_1) #15`	`sum(set_1) #15`

Find min/max	`min(list_1) #1`
	`max(list_1) #5`
		`min(tuple_1) #1`
		`max(tuple_1) #5`
			`min(set_1) #1`
			`max(set_1) #5`

Add more element	`list_1.append(6)`
	`#[1,2,3,4,5,6]`
		`Tuple objects are immutable`	`set_1.add(6)`
			`#{1,2,3,4,5,6}`

Additional operations	`del list_1[2] #[1,2,4,5,6]`
	
	`#Remove last item in a list`
	`list.1.pop() #[1,2,4,5]`
	
	`#Sorting with replacement`
	`list.1.sort() #[1,2,4,5]`
	
	`#Sorting without replacement`
	`Sorted(list_1) #[1,2,4,5]`
			`set_2 = {3, 4}`
			
			`#union #{1,2,3,4,5,6}`
			`set_1 | set_2`
			`set_1.union(set_2)`
			
			`#Intersection #{3,4}`
			`set_1 & set_2`
			`set_1.intersection(set_2)`
			
			`#Difference #{1,2,5,6}`
			`set_1 - set_2`
			`set_1.difference(set_2)`
			
			`#Symmetric Difference #{1,2,5,6}`
			`set_1 ^ set_2`

Importance	◦ Slower in performing iterations than a tuple ◦ Has many built-in methods ◦ Takes more memory as compared to tuple ◦ Better for operations, such as insertion and deletion	◦ Consume less memory but has very less in-built methods ◦ Implications are comparatively faster than the list ◦ Better than the list for accessing elements	◦ Significantly faster in checking if an element is present then lists or tuples ◦ Slower than list or tuples when performing iterations

Introduction to Python Programming

Dictionaries: key:value relationship

```
dictionary = {'john':'Professor','Martin':'Engineer'}
#Change value of an existing key
dictionary['John']='Doctor'
#{'John':'Doctor','Martin':'Engineer'}
del dictionary['John'] # {'Martin':'Engineer'}
dictionary['James'] = 'Scientist'
dictionary.update({key:value}) #Add a key:value pair
dictionary.keys() #Retrieve all keys
dictionary.values() #Retrieve all values
```

Conditional statement:

If-else statement: To execute the true or the false part of a given condition

If-elif-else statement: Used for decision-making

```
a = float(input())
if (a <= 20):
   print ('a is less or equal to 20')
elif (a>20 and a <= 30):
   print ('a is in between 20 and 30')
elif (a>30 and a <= 40):
   print ('a is in between 30 and 40')
else:
   print ('a is greater than 40')
```

Basis of Comparison	For loop	While loop
Declaration	for object in sequence: <indentation>body	while condition: <indentation>body
Use Case	When you already know the number of iterations	When you don't know the number of iteration.
Iteration	Every element is fetched through the iterator/generator. Example, range()	Every element is explicitly incremented or decremented by the user
Generator Support	For loop can be iterated on generators in Python	While loop cannot be iterated on generators directly
Generator Support	For loop can be iterated on generators in Python	While loop cannot be iterated on generators directly
Speed of execution of code	For loop is faster than while loop	While loop is slower than for loop

Example:
Finding factorial
	`num = int(input('Integer value:'))`
	`if num>0:`
	`    fac = 1`
	`    for i in range(1, num + 1):`
	`    fac = fac * i`
	`    print('Answer', num)`
	`elif num<0:`
	`    print('Invalid input')`
	`else:`
	`    print('Zero factorial is 1')`
		`num = int(input('Integer value:'))`
		`if num>0:`
		`    fac = 1`
		`    i = 1`
		`    while i <= num:`
		`        fac = fac * i`
		`        i = i + 1`
		`    print('Answer:', fac)`
		`elif num<0:`
		`    print('Invalid input')`
		`else:`
		`    print('Zero factorial is 1')`

#The following methods are executed along with the for loop to iterate over each element in a list, set or dictionary

#List comprehension

```
#Check if number is divisible by 2
[x for x in range (5) if x%2 == 0]
#[0, 4, 16]
```

#Dictionary comprehension

```
#Length of words
words = ['data', 'science']
{i: len(i) for i in words}
#{'data':4, 'science': 7}
```

#Set comprehension

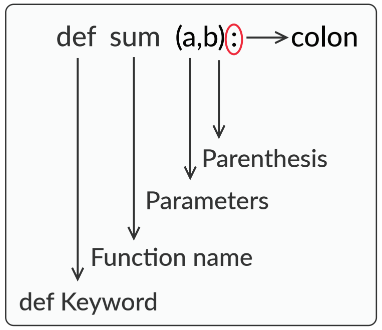
```
#Multipying each number by 3
set1 = {1,2,3,4}
set1 = {i*3 for i in set1}
```

Introduction to Python Programming

Function

```
#Square of any given number:
k = float(input())
def square(k):
  return k**2

square(k)
```



map()

```
items = [1,2,3,4,5]
square = lambda x: x**2
list (map(square, items))
#[1, 4, 9, 16, 25]
```

filter()

```
list(filter((lambda x: x < 0),range(-5,5)))
#[-5, -4, -3, -2, -1]
```

Lambda Function

- Used to define a single expression with any number of arguments
- Helpful when the function is used for 1 or 2 instances

```
square = lambda x: x**2  #square(10) #100
```

reduce()

```
#Find factorial:
from functools import reduce
n = int(input())
if n>0:
    print('Factorial of',n,'is',reduce((lambda x, y:x * y),range(1,n+1)))
elif n==0:
    print (1)
else:
    print('Factorial is not defined on negative integer')
```

Range: Print all integers in a certain interval

Syntax: range(start, stop, step)

```
#all integers from 0(included) upto 10(excluded)
range(0,10)
#print all elements of the range in list format
list(range(0,10) #[0,1,2,3,4,5,6,7,8,9]
list(range(10,0,-2)) #[10,8,6,4,2]
```