Comprehensions

It provides a concise way to create lists, dictionaries and sets (generate data)

1

List Comprehension:

provide a concise way to create lists. They can also include conditions to filter items.

SYNTAX

expression **for_loop** condition

EXAMPLE

zx = [i for i in range(1,21) if i%2 == 0]

2

Set Comprehension:

similar to list comprehensions but create a set (which automatically removes duplicates).

SYNTAX

expression **for_loop** condition

EXAMPLE

xz = {f"2*{i} = {2*i}" for i in range(1,11)}

3

Dictionary Comprehension:

It allow to construct dictionaries with key-value pairs.

SYNTAX

key_expression : value_expression **for_loop** condition

EXAMPLE

y = {i: i**2 for x in range(1, 6)}

Made with Gamma

Lambda Functions/Anonymous Function

These functions are ideal for creating small and anonymous functions for minor tasks

SYNTAX

lambda parameter : expression

EXAMPLE

```
zx = lambda a : a*2

print(zx(2))
```

```
xy = lambda a,b : print(a+b)

xy(2,5)
```

Use Cases

map()

Apply function to each element

filter()

filter elements based on condition

reduce()

Cumulatively applies the function on elements to reduce iterable to a single value

1

map()

SYNTAX

map(lambda_function, iterator)

EXAMPLE

```
zx = [2,3,5,1,4]
pq = map(lambda i:i*i, zx)
print(list(pq))
```

2

filter()

SYNTAX

filter(lambda_function, iterator)

EXAMPLE

```
zx = [2,3,5,1,4,6,10,23,12]
pq = filter(lambda x:x%2 == 0, zx)
print(list(pq))
```

3

reduce()

SYNTAX

reduce(lambda_function, iterator)

EXAMPLE

```
from functools import reduce
zx = [2,3,5,1,4,6,10,23,12]
pq = reduce(lambda a,b:a+b, zx)
print(pq)
```

OOP'S (Object Oriented Programming)

- **Class** (blueprint of an object)
- **Object** (instance of a class)
- **Constructor** (special method in class that automatically called itself when an object of that class is created) (built using **__init__**)
- **Self** (predefined parameter help to referencing)
- **Attributes** (variable that holds data associated with an object or class) (modified and access directly)
- **Methods** (in oops fxn.s called methods)
- **Inheritance** (allows a new class to be based on an existing class and access one class property to another class)
  Single Inheritance, Multiple Inheritance, Multilevel Inheritance, Hierarchal Inheritance, Hybrid Inheritance
- **Abstraction** (exposing only necessary features of object while hiding the implementation details.)
  Abstract Class, Abstract Method, Interface
- **Encapsulation** (it is the concept of bundling multiple things(data & methods) within a single unit(class)
  Public Member, Protected Member (**_** use to create), **Private Member** (**__** use to create)(
- **Polymorphism** (many forms)(it is the ability of objects to take on multiple forms(same object have different behavior))
  Runtime Polymorphism——Method Overriding, Compile time Polymorphism—— Operator Overloading and Method Overloading