Introduction to Flow Control

Flow control refers to the order in which individual statements, instructions, or function calls are executed or evaluated in a programming language. Python's primary flow control structures are `if`, `for`, and `while`.

Example Scenario

Scenario: Imagine you are writing a program for a library system where you need to check if a book is available for borrowing.

Using the `if` Statements

Conditions allow you to execute certain pieces of code based on whether a condition is true or false. The `if` statement is used to test a condition and execute a block of code if the condition is true.

In [ ]:
```
book=input("Enter book Name:")

is_GOF_avail=True

if book=="Goblet of Fire":
    if is_GOF_avail==True:
        print("The book is available for borrowing.")
else:
    print("The book is currently unavailable.")
```

```
The book is currently unavailable.
```

Using `if`, `else`, `elif` Together.

`If` is used when you want to ask a question. `else` is used when you have another condition when the If block does'nt satisfy/

`elif` is used when you have multiple conditions.

In [ ]:
```
copies_available=2

#

if copies_available == 2:
    print("Two copies are available.")
elif copies_available == 1:
    print("Only one copy is available.")
elif copies_available == 0:
    print("No copies are available.")
else:
    print(f"{copies_available} copies are available.")
```

```
Two copies are available.
```

Conditions

`for` Loops

A `for` loop is used for iterating over a sequence (e.g., a list, a tuple, a dictionary, a set, or a string).

In [ ]:
```
books = ["Goblet Of Fire", "Prisoner Of Azkaban", "Sorcerers Stone", "Order Of TRhe Phoenix", "Deathly Hallows"]

# 1st Way
# for i in books:
#     print(i)

#2nd way
for i in range(0,len(books)):
    print(books[i])

#3rd way
for index,book in enumerate(books):
    print(index, book)
```

```
# # #Advanced
# import itertools

# # # for i in range(0,len(books)):
# # #     0--4

# for x in itertools.count(0,2):
#     if x==len(books):
#        break
#     print(books[x])

# #     # if x>len(books):
# #     #     break
# #     # else:
# #     #     print(books[x])
```

```
0 Goblet Of Fire
1 Prisoner Of Azkaban
2 Sorcerers Stone
3 Order Of TRhe Phoenix
4 Deathly Hallows
```

In [ ]:
```
for index,book in enumerate(books):
    print(index, book)

print(books[2])
```

```
0 Goblet Of Fire
1 Prisoner Of Azkaban
2 Sorcerers Stone
3 Order Of TRhe Phoenix
4 Deathly Hallows
Sorcerers Stone
```

`while` Loops

A `while` loop repeats as long as a condition is true.

In [ ]:
```
books = ["Goblet Of Fire", "Prisoner Of Azkaban", "Sorcerers Stone", "Order Of TRhe Phoenix", "Deathly Hallows"]
```

```
count=4
while count>0:
    book_title = input("Enter a book title (type 'exit' to stop): ")
    if book_title in books:
        print('Yes its Available!')
        count=count-1
        continue
    else:
        print('Nope Its not Available')
        break
```

```
Yes its Available!
Yes its Available!
Nope Its not Available
```

In [ ]:
```
#List

#Time Complexity: o(n) n-->no of data i have

#Dictionary

#Time complexity : o(1)
```

Introduction to Dictionaries

A dictionary is a collection of key-value pairs. It is unordered, changeable, and indexed. Each key is unique. A dictionary in Python is a data structure that stores data in key-value pairs. It is similar to a real-world dictionary where you have words (keys) and their definitions (values). Each key in a dictionary is unique, and it maps to a specific value. This allows for efficient data retrieval because you can access the value associated with a key very quickly, just like looking up the definition of a word in a dictionary.

Key Features:

`Key-Value Pairing` : Each item in a dictionary has a unique key and a corresponding value.

`Unordered` : Unlike lists or tuples, dictionaries are unordered. This means the items are not stored in a specific sequence.

`Mutable` : You can change, add, or remove items after the dictionary has been created.

`Fast Access` : Retrieving a value using its key is very fast, almost instantaneous, because dictionaries use a hash table internally. Real-World Analogy

Dictionary Methdos:

`Add Key-Value Pairing`

`Remove Key-Value Pairing`

`Update Key-Value Pairing`

Adding Books to library

In [ ]:
```
library_dictionary={}

#Method 1
# library_dictionary["Goblet of Fire"] = True
# library_dictionary["Prisoner of Azkaban"] = True
# library_dictionary["Sorcerer's Stone"] = True
# library_dictionary["Order of the Phoenix"] = True
# library_dictionary["Deathly Hallows"] = True



#Method 2
# for book in books:
#     library_dictionary[book] = True


# #Method 3
# val = True
# thisdict = dict.fromkeys(books, val)
# print(thisdict)
```

Accessing and Iterating A Dictionary

In [ ]:
```
#Access Values
#spec_key=''
```

```
# keys= list(library_dictionary.keys())
# print(keys)

# values= list(library_dictionary.values())
# print(values)
# # #specific=library_dictionary[f'']

# # #Iterating
# for key,val in library_dictionary.items():
#     print (key,val)
```

```
Goblet Of Fire True
Prisoner Of Azkaban True
Sorcerers Stone True
Order Of TRhe Phoenix True
Deathly Hallows True
```

Updating Dictionary

In [ ]:
```
library_dictionary.update({"Chamber of Secrets" : "True"})
```

In [ ]:
```
library_dictionary
```

Out[ ]:
```
{'Goblet Of Fire': 'False',
 'Prisoner Of Azkaban': True,
 'Sorcerers Stone': True,
 'Order Of TRhe Phoenix': True,
 'Deathly Hallows': True,
 'Chamber of Secrets': 'True'}
```

Deleting Dictionary

In [ ]:
```
#method 1
del library_dictionary["Goblet Of Fire"]

#method 2
library_dictionary.pop("Deathly Hallows")
```

Out[ ]:
```
True
```

In [ ]:
```
library_dictionary
```

Out[ ]:
```
{'Prisoner Of Azkaban': True,
 'Sorcerers Stone': True,
 'Order Of TRhe Phoenix': True,
 'Chamber of Secrets': 'True'}
```

Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target.

You may assume that each input would have exactly one solution, and you may not use the same element twice.

You can return the answer in any order.

Input: nums = [2,7,11,15], target = 9

Output: [0,1]

Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].

In [ ]:
```
nums = [2,7,11,15]
target = 9

seen={}

for i, num in enumerate(nums):
    print(target-num)
    if target-num in seen:
        print([seen[target-num],i])
    elif num not in seen:
        seen[num]=i

seen
```

In [ ]:
```

```