



CS 200I PYTHON PROGRAMMING



2 LECTURE OBJECTIVES

- Introduce the concept of 'container types' in python
- Explore the characteristics of 'list' in python

3 CONTAINER TYPE

- It refers to a collection of data or objects- Can be considered as derived datatypes
- Objects can be of similar type or of different.
- Python has a wide variety of container types
- Four different container types are directly available with python
 - List, tuple, set, dictionary
- Many other container types are available as part of the python module “collections”
 - namedtuple(), ordereddict(), deque etc

4 COLLECTIONS

<code>namedtuple()</code>	factory function for creating tuple subclasses with named fields
<code>deque</code>	list-like container with fast appends and pops on either end
<code>ChainMap</code>	dict-like class for creating a single view of multiple mappings
<code>Counter</code>	dict subclass for counting <code>hashable</code> objects
<code>OrderedDict</code>	dict subclass that remembers the order entries were added
<code>defaultdict</code>	dict subclass that calls a factory function to supply missing values
<code>UserDict</code>	wrapper around dictionary objects for easier dict subclassing
<code>UserList</code>	wrapper around list objects for easier list subclassing
<code>UserString</code>	wrapper around string objects for easier string subclassing

5 PROPERTIES OF CONTAINER TYPES

- List – iterable, mutable, ordered collection
- Tuple – iterable, immutable, ordered collection
- Set – iterable, mutable, unordered collection
- Dictionary – iterable, mutable, unordered collection

6 SYNTAX

- `List l = [1,2,3,4,5]`
- `Tuple l = 1,2,3,4,5` or `Tuple l = (1,2,3,4,5)`
- `Set l = {1,2,3,4,5}`
- `Dict l = {CS2001:"PYTHON",CS2003:"COA",CS2005:"TOC"}`

7 LIST

- Primary container type in python
- Iterable, mutable, ordered collection of data
- Can include dissimilar members but normally used for the collection of similar type objects
- Can be instantiated in two ways
 - By enclosing list elements within []
 - By using built in function list()
 - The function list() can convert a tuple , a string or a set into list

8 ACCESS OF LIST ELEMENTS

- Details of the list can be displayed by typing `print(list name)`
- `LI = ['Messi', 10, 'FIFA', 1]`
- Eg `print(LI)` will display the whole list as `['Messi', 10, 'FIFA', 1]`
- Element by element access is possible in the case of list
- Indexing starts from 0
- `print(LI[0])` will display 'Messi'
- Slicing is also possible in list
- `Print(LI[0:2])` will display `['Messi', 10]`

9 MUTABILITY

- Unlike strings, list is a mutable object
- `L1=[1,2,3,4]`
- `print(L1)`
- `L1[0]=10`
- `print(L1)`

10 ITERATIONS USING LIST

- List can be used in a loop for definite number of iterations

- while i < len(Listl):

```
    print("Iteration" ,i,"using List")
```

```
    i=i+1
```

```
for j in Listl:
```

```
    print("Iteration using List")
```

II ITERATION USING ENUMERATE()

- To keep track of the index during iteration , python provides built in function `enumerate()`

```
Fruits=['apple','orange','mango','guava']
```

```
for index, item in enumerate(Fruits):
```

```
    print(index, item)
```

12 SPLIT() AND JOIN()

- A string can be split() to form a list
- List items can be joined to form a string

```
S1='India is my country'
```

```
type(S1)
```

```
L1=S1.split()
```

```
type(L1)
```

```
L2=L1[::-1]
```

```
S2=' '.join(L2)
```

```
print(S2)
```

I3 BASIC OPERATIONS ON LISTS

- Concatenation
- Replication
- Aliasing
- Cloning
- Searching
- Identity
- Comparison
- Emptiness

14 CONCATENATION

- Similar to that of strings
- '+' is the operator
- $L1=[1,2,3,4,5]$
- $L2=[5,4,3,2,1]$
- $L3=L1+L2$

```
print(L1,'\n',L2,'\n',L3)
```


15 REPLICATION

- `L1=[1,2,3]`
- `L2=L1*3`
- `print(L2)`

OUTPUT

`[1,2,3,1,2,3,1,2,3]`

16 ALIASING

- Using '=' operator, one list can be assigned to another variable.
- Both the variables will refer to the same list.
- This is called shallow copying or aliasing

```
L1=[1,2,3,4,5]
```

```
L2=L1
```

```
print(L1,'\n',L2)
```

```
print(id(L1))
```

```
print(id(L2))
```

17 CLONING

- Copying the content of one list to another list
- In this case, two objects will be created
- The lists will refer to two different objects.
- Modification in one list will not affect the other.
- `L1=[1,2,3,4,5]`
- `L2=[]`
- `L2=L2+L1` or `L2[:]=L1` (Both are valid statements)
- `L1[1]=6`
- `print(L1,'\n',L2)`

18 SEARCHING

- We can use membership operator to search for a particular item

```
List1=['s','e','a','r','c','h','i','n','g']
```

```
Bool1='a' in List1
```

```
Bool2 = 'b' in List1
```

```
Bool3 = 'c' not in List1
```

19 IDENTITY

- Using 'is' operator

`L1=L2`

`Res=L1 is L2`

`print(Res)`

20 COMPARISON

- Using comparison operators
- '>', '<'
- Comparison is done item by item and returns the result of first mismatch
- Returns True or False
- `A=[1,2,3,9,5]`
- `B=[1,2,6,7,8]`
- `Res1 = A<B`
- `Print(Res1)`

21 EMPTINESS

- Can be checked using not operator or bool function
- Empty list is treated as False by Python

```
L1=[1,2,3,4]
```

```
L2=[]
```

```
print(bool(L1))
```

```
print(bool(L2))
```

Using 'not' operator

```
if not L1:
```

```
    print("Empty List")
```

22 FUNCTIONS FOR LIST (PASSING LIST OBJECT)

- `len(list)` – to find the number of elements present in a list
- `max(list)` – to find the maximum value of a list
- `min(list)` – to find the minimum
- `sum()` – to find the sum of all elements
- `any()` – returns True if any of the list element is True
- `all()` – returns True if all of the list elements are True
- `del()` – to delete a complete list object or any item of the list from the memory
- `sorted()` – to create a new list object which is a sorted version of the original list

23 NESTED LISTS

- We can include one list as a member of another list

```
LI=[1,3,[11,13,15,[17,19],21],5,7]
```

```
print(LI)
```

```
print(LI[0:3])
```

```
print(LI[2])
```

```
print(LI[2][0])
```

```
Print(LI[2][3])
```

```
print(LI[2][3][0])
```

24 UNPACKING A STRING OR LIST

- `S="UNPACK"`
- `S1=[*S]`
- `print(S1)`
- `L1=[1,2,5]`
- `L2=[7,8,9,L1,10]`
- `print(L2)`
- `L3=[7,8,9,*L1,10]`
- `print(L3)`

25 METHODS FOR LIST

- `append()` – to add an item at the end of the list
- `sort()` – to sort the list in place
- `copy()` – shallow copying
- `clear()` – to empty the list
- `extend()` – to extend the list with another object
- `remove()` – to remove the specified item
- `pop()` – to remove the last element (`pop()` can be used to remove an item at a specified index)
- `insert()` – to insert an item at a specified index
- `Index()`- to get the position of a particular element in the list

26 LIST COMPREHENSION

- Feature provided by python to generate a list with a single line of code according to the requirements
- Syntax
- [expression or variable<space> for loop <space> zero or more for loop or if clause].
- Comprehension helps to reduce the number of lines in the program
- There are various code segments which can be replaced by list comprehension
- It helps to store the output in the form of a list object

27 COMPREHENSION WITH SINGLE FOR LOOP

- Generating square of first n natural numbers

`Nsquares=[j*j for j in range(1,n)]`

- Generating a list of first n natural numbers, their squares and their cubes

`Nlist=[[j,j*j,j*j*j] for j in range(1,n)]`

- Convert a list of strings into a list of integers

`IntList=[int(n) for n in ['10','15','20','25']`

28 COMPREHENSION WITH FOR AND IF

- Generating a list of all even numbers within the range of 20 to 50

```
Even=[n for n in range(20,50) if n%2==0]
```

```
print(Even)
```

- Generating a list of students whose name starts with the letter 'A'

```
StudentsList=["Aditya", "Shraddha", "Madhuri", "Sruthi", "Samridhi", "Anmol", "Titiksha", "Praveen"]
```

```
Newlist=[Name for Name in StudentList if Name.startswith('S')]
```

```
print(Newlist)
```

29 COMPREHENSION WITH IF AND ELSE

- In this case, if clause and else will be written before for loop
- To generate $n*n$ if $n>5$ and $n*n*n$ if $n\leq 5$

ListN=[$n*n$ if $n>5$ else $n*n*n$ for n in range(-10,10)]

- To replace all the vowels in a string with '*'

StringNew=['*' if letter in 'aeiouAEIOU' else letter for letter in 'Education']

30 COMPREHENSION WITH NESTED FOR

- One for loop followed by another for loop within the comprehension
- Flattening a list of lists
- `List = [[1,2,3],[4,5,6],[7,8,9]]`
- `L=[n for row in List for n in row]`
- `print(L)`

3 | NESTED COMPREHENSION

- Like nested for, comprehensions can also be nested
- See the difference between the output of the statements
- `List_P1=[i*j for i in range(1,5) for j in range(1,5)]` and
- `List_P2=[[i*j for i in range(1,5)] for j in range(1,5)]`

32 ENTERING MULTIPLE INPUTS

- To instantiate multiple integer objects at run time
- `Nlist= [int(n) for n in input().split()]`
- It is also possible to create a list of strings
- `Namelist=[name for name in input().split()]`
- In both the cases, the length of the list is determined at run time

Thank You





CS 200I PYTHON PROGRAMMING



2 LECTURE OBJECTIVES

- Introduce the concept of 'set' in python

3 SET

- Container type in python that follows the concept of 'set' in Mathematics
- Mutable, iterable, Unordered collection of data
- Can include dissimilar members
- Can be instantiated in two ways
- By enclosing set members in {}
 - `S1 = {"Messi", 10, "FIFA", 123}`
- By using built in function `set()` – it can convert a list or a tuple into a set
 - `S2 = set(("Messi", 10, "FIFA", 123))`
 - `S3 = set([1, 2, 3, 4])`

4 PROPERTIES OF SET ELEMENTS

- Details of the set can be displayed by typing `print(lset name)`
- **Element by element access is not possible in the case of set**
- **Duplication of items is not possible in set**
- **Each item will have a unique hash value**
- Only hashable(immutable objects) can become the members of a set
- Basic datatypes like int, float, bool, strings and tuple can be included in a set

5 ITERATIONS OF SET

- Set items can be displayed using for loop but not with while loop
- for j in S1:
 print(j)
- To keep track of the index during iteration , python provides built in function enumerate()

Fruits={'apple','orange','mango','guava'}

for index, item in enumerate(Fruits):

 print(index, item)

Since set is an unordered collection, the order of display or index is not in user's control

6 NESTED SETS

- We cannot include a set as a member of another set
- Only immutable objects can be included as items of a set
- We can include a string or tuple as a member of a set
- A list being mutable cannot be included in a set

7 MUTABILITY

- Since `[]` operator is invalid, modification of an existing set is possible with the help of built in functions or certain update operators
- Any attempt for slicing or indexing a set will raise an error in Python

8 BASIC OPERATIONS ON SETS

- Aliasing
- Cloning
- Searching
- Identity
- Comparison
- Emptiness

9 ALIASING

- Using '=' operator
- $S1 = \{1, 2, 3, 4\}$
- $S2 = S1$
- $id(S1)$
- $id(S2)$

10 CLONING

- $S1 = \{1, 2, 3, 4\}$
- $S2 = \text{set}()$
- $S2 = S2 + S1 \rightarrow$ see the error

Cloning can be done using built in method `copy()`

II SEARCHING

- We can use membership operator to search for a particular item

Set1={'s','e','a','r','c','h','i','n','g'}

Bool1='a' in Set1

Bool2 = 'b' in Set1

Bool3 = 'c' not in Set1

I2 IDENTITY

- Using is operator

S1=S2

Res=S1 is S2

print(Res)

I3 COMPARISON

- Using == , > , < , >= , <= operators
- Returns True or False
- $A = \{1, 2, 3, 9, 5\}$
- $B = \{1, 2, 6, 7, 8\}$
- $A > B$ returns True if A is a superset of B
- $A < B$ returns True if A is a subset of B
- $A == B$ returns True if the contents of A and B are identical

14 EMPTINESS

- Can be checked using not operator or bool function

```
S1={1,2,3,4}
```

```
S2=set()
```

```
print(bool(S1))
```

```
print(bool(S2))
```

Using 'not' operator

if not S1:

print("Empty Set")

I5 FUNCTIONS FOR SET(PASSING SET OBJECT)

- `len()`
- `max()`
- `min()`
- `sum()`
- `any()`
- `all()`
- `del()` – Here `del()` can be used to delete a set object (Slicing is not possible in set)
- `sorted()` – It will generate the sorted list and not a set

16 METHODS FOR SET

- `copy()` – deep copy
- `clear()` – to clear the contents of a set (to create an empty set)
- `add()` – to add any item or object
- `remove()` – to remove certain item
- `pop()` – to remove a specific item (Since the order of the set is unknown, the element to be removed by `pop()` in set is not user's choice)
- `discard()` – To discard certain item

17 MATHEMATICAL SET OPERATIONS

- Union – operator ‘|’
- Intersection – operator ‘&’
- Difference – operator ‘-’
- Symmetrical Difference – ‘^’
- All these operators will generate new set objects
- These operations can be done with the help of set methods
 - `Set.union()`, `Set.intersection`, `Set.difference` , `Set.symmetrical_difference`

18 SET UPDATE - METHODS AND OPERATORS

- To modify the existing set, methods available are
 - `Set1.update(Set2)`
 - `Set1.intersection_update(Set2)`
 - `Set1.difference_update(Set2)`
 - `Set1.symmetric_difference_update(Set2)`
- Operators available
 - `Set1 |= Set2`
 - `Set1 &= Set2`
 - `Set1 -= Set2`
 - `Set1 ^= Set2`

19 SET METHODS FOR RELATING TWO SETS

- `Set1.issuperset(Set2)`
 - Returns True if Set1 is a superset of Set2
- `Set1.issubset(Set2)`
 - Returns True if Set1 is a subset of Set2
- `Set1.isdisjoint(Set2)`
 - Returns True if Set1 and Set2 are mutually exclusive

20 SET COMPREHENSION

- Similar to that of list comprehension but has to comply with the validity of sets
- Case 1: Single for loop
- $\{n+1 \text{ for } n \text{ in range}(1,10)\}$
- Try $\{(n,n*n) \text{ for } n \text{ in range}(1,10)\}$
- Try $\{[n,n*n] \text{ for } n \text{ in range}(1,10)\}$
- Try $\{(n,n*n) \text{ for } n \text{ in range}(1,10)\}$

21 SET COMPREHENSION

- Case 2: for loop with if
- This will help to generate a set for a given range when the condition given in if is True
- Example – generate a set of numbers that are divisible by 6 within the range of 0 to 100
- $\{n \text{ for } n \text{ in range}(0, 100) \text{ if } n \% 6 == 0\}$

22 SET COMPREHENSION

- Case 3: for with if and else
- In this case, if clause and else will be written before for loop

To generate $n*n$ if $n>5$ and $n*n*n$ if $n\leq 5$

- $\text{SetN}=\{n*n \text{ if } n>5 \text{ else } n*n*n \text{ for } n \text{ in range}(-10,10)\}$
- The condition should not lead to duplication of elements
- Try $\text{setI}=\{n**0.5 \text{ if } n<6 \text{ else } n//4 \text{ for } n \text{ in range}(0,20)\}$

23 SET COMPREHENSION

- Case 4: Nested for – Try yourself
- Case 5: Multiple input – Try yourself
- Case 6 : Nested comprehension which is not applicable in the case of set

Thank You





CS 200I PYTHON PROGRAMMING

2 LECTURE OBJECTIVES

- Explore the characteristics of 'tuple' in python

3 TUPLE

- Primary container type in python
- Iterable, immutable, ordered collection of data
- Meaning of tuple – a structure of data that has several parts
- Tuple in python is usually for storing dissimilar data
- Can be instantiated in two ways
 - By enclosing elements within () or without parenthesis separated by comma
 - By using built in function tuple()
 - The function tuple() can convert a list , a string or a set into tuple

4 ACCESS OF TUPLE ELEMENTS

- Details of the tuple can be displayed by typing `print(tuple name)`
- `TI = ('Messi', 10, 'FIFA', 1)`
- Eg `print(TI)` will display the whole list as `('Messi', 10, 'FIFA', 1)`
- Element by element access is possible in the case of tuple
- Positive Indexing starts from zero and negative indexing starts from -1
- `print(TI[0])` will display 'Messi'
- Slicing is also possible in tuple
- `Print(TI[0:2])` will display `('Messi', 10)`

5 ITERATIONS USING TUPLE

- Tuple can be used in a loop for definite number of iterations

- while i < len(T1):

```
    print("Iteration" ,i,"using T1")
```

```
    i=i+1
```

```
    print(T1[i])
```

```
for j in T1:
```

```
    print("Iteration using tuple")
```

```
    print(j)
```


6 ITERATION USING ENUMERATE()

- To keep track of the index during iteration , python provides built in function `enumerate()`

```
Fruits=('apple','orange','mango','guava')
```

```
for index, item in enumerate(Fruits):
```

```
    print(index, item)
```

7 SWAPPING OF DATA

- Tuple assignment can be used to swap between the data assigned to variables
- `a,b=20,30`
- `a,b=b,a`
- `print(a,b)`
`a,b,c=10,20,30`
- `a,b,c=c,a,b`
- `print(a,b,c)`

8 MUTABILITY

- Unlike list, tuple is immutable
- `T1=(1,2,3,4)`
- `print(T1)`
- `T1[0]=10` → See the error
- But tuple can include unhashable (mutable) objects
- Tuple allows duplication of elements

9 MUTABILITY

- Tuple can have list as its member and the list within a tuple can be modified

```
L1=[1,2,3]
```

```
T1=1,2,L1,4
```

```
print(T1)
```

```
L1[0]=5
```

```
print(T1)
```

10 NESTED TUPLES

- We can have tuple of tuples, tuple of lists, tuple of sets etc
- `T1=1,2,3`
- `T2=4,5,6`
- `T3=7,8,9`
- `T=T1,T2,T3`
- `print(T)`
- `T[0][0]`

II UNPACKING A TUPLE

- Using * operator
- T1=1,2,3
- T2=6,7,T1,8
- print(T2)
- T2=6,7,*T1,8
- print(T2)

I2 LIST OF TUPLES, TUPLE OF LISTS ...

- $LT = [(1,2,3), (4,5,6), (7,8,9)]$
- $TL = ([1,2,3], [4,5,6], [7,8,9])$
- $ST = \{(1,2,3), (4,5,6), (7,8,9)\}$
- $TS = \{1,2,3\}, \{4,5,6\}, \{7,8,9\}$

I3 ZIP FUNCTION

- To incorporate items of different iterable objects into a single container type
- Mostly to generate a collection of dissimilar data
- zip can create a list of tuples, tuple of tuples, set of tuples etc
- Example:
- `SI="ABCD"`
- `LI=[1,2,3,4]`
- `TI=(5,6,7,8)`
- `L=list(zip(SI, TI, LI))`

I4 PARTITION AND JOIN

- S1="Hadhi mera Sadhi"
- T1= S.partition(" ")
- print(T1)
- S2=" ".join(T1)
- print(S2)

15 BASIC OPERATIONS ON TUPLE

- Concatenation
- Replication
- Aliasing
- Cloning
- Searching
- Identity
- Comparison
- Emptiness

16 COMPARISON

- Using comparison operators
- '>', '<'
- Comparison is done item by item and returns the result of first mismatch
- Returns True or False
- A=1,2,3,9,5
- B=1,2,6,7,8
- Res1 = A<B
- print(Res1)

17 FUNCTIONS FOR TUPLE (PASSING TUPLE OBJECT)

- `len()` – to find the number of elements present in a tuple
- `max()` – to find the maximum value of a tuple
- `min()` – to find the minimum
- `sum()` – to find the sum of all elements
- `any()` – returns True if any of the tuple element is True
- `all()` – returns True if all of the tuple elements are True
- `del()` – to delete a complete tuple object
- `sorted()` – to create a new list object which is a sorted version of the original tuple

18 METHODS FOR TUPLE

- `index()`- to get the position of a particular element in the tuple
- `count()` – to get the number of occurrences of a particular item
- Being immutable, other methods and comprehension are not available for tuple

Thank You





CS 200I PYTHON PROGRAMMING

2 LECTURE OBJECTIVES

- Introduce the concept of 'dictionary' in python

3 DICTIONARY

- Container type in python that follows the concept of 'dictionary'
- Mutable and iterable
- Older versions of python treats dictionary as an unordered collection
- Python 3.7 onwards dictionary preserves insertion order but not position indexable
- In dictionary, each item is a key value pair
- Dictionary elements are key indexable
- Only hashable objects can be used to make the keys of a dictionary
- Can include dissimilar members

4 INSTANTIATION OF DICTIONARY

- `D1={'a':20,'b':30,'c':40}`
- `print(D1)`
- `print(type(D1))`
- `D2={1:20,2:30,3:40}`
- `print(D2)`
- `D3=dict(a=20,b=30,c=40) → see D3`
- `D4=dict(1=20,2=30,3=40) → see the error`

5 PROPERTIES OF DICTIONARY

- **Element by element access using [position index] is not possible in dictionary**
- **Values are accessed using the key index**
- **Duplication of key is not possible in dictionary**
- **Each key should have a unique hash value**
- Only hashable(immutable objects) can become the key
- Basic datatypes like int, float, strings and tuple can be assigned as keys
- Value associated with key can be of any type(hashable or unhashable)

6 ITERATIONS OF DICTIONARY

- Dictionary items can be displayed using for loop and can be done different ways

- for j in D1.keys():

```
    print(j)
```

```
for j in D1.values():
```

```
    print(j)
```

```
for k in D1:
```

```
    print(k)
```

```
for k in D1:
```

```
    print(D1[k])
```

```
for k,v in D1.items():
```

```
    print(k,v)
```

7 MUTABILITY

- Dictionary is mutable – can modify, add and delete the items
- The existing value can be modified using its key
- Captains={"India":"Rohit","Australia":"Cummins","Pakistan":"Babar"}
- print(Captains)
- Captains["Australia"]="Smith"
- print(Captains)

8 MUTABILITY

- New item can be added by adding a new key and its associated value
- `Captains["NewZeland"]="Williamson"`
- `print(Captains)`
- The existing item can be deleted by deleting the key
- `del(Captains["India"])`
- `print(Captains)`

9 SLICING

- Slicing of a dictionary is not possible
- This is due to key indexing
- Try `Captains["Australia":]` or `Captains[:]"NewZeland"]`

10 NESTED DICTIONARY

- We can have a dictionary of dictionaries
- `Student1=dict(Name="Madhuri", RollNo=1001, Marks=75)`
- `Student2=dict(Name="Ritik", RollNo=1074, Marks=80)`
- `Student3=dict(Name="Devam", RollNo=3016, Marks=85)`
- `D1=dict(I=Student1, II=Student2, III=Student3)`
- `print(D1)`

II NESTED DICTIONARY

- Nested dictionary elements can be iterated over a loop and can make collection of various items present in the dictionary.

```
NameList=[]
```

```
for student in DI.values():
```

```
    print(student["Name"])
```

```
    NameList.append(student["Marks"])
```

-

12 UNPACKING A DICTIONARY

- We can unpack the keys of a dictionary using * operator
- FruitsPrice={"Mango":50,"Apple":100,"Guava":30}
- *FruitsPrice → See the error
- [*FruitsPrice] → view the list of keys
- {*FruitsPrice} → view the set of keys

13 MERGING USING **

- Two dictionaries whose keys are different can be merged using ** operator
- `FruitsPrice={"Mango":50, "Apple":100, "Guava":30}`
- `VeggiePrice={"Onion":25, "Potato":30, "Tomato":60}`
- `Pricelist={**FruitsPrice, **VeggiePrice}`
- `print(Pricelist)`
- If we try to merge two dictionaries with same keys, only the latter will be inserted.
(Whenever there is duplication of keys, values of the last group of keys are taken)

14 BASIC OPERATIONS ON DICTIONARIES

- Concatenation – will not work using + operator but can do merging using **
- Aliasing – does exist
- Searching – can check whether a particular key is present in that dictionary
- Identity – does exist
- Comparison – does not exist
- Emptiness – {} indicates empty dictionary and is treated as False

15 ALIASING

- Using '=' operator
- Student4=Student1
- id(Student1)
- id(Student4)

16 CLONING

- `Student5={}`
- `Student5=Student5+Student1` → error

`Student5={**Student1}`

17 SEARCHING

- We can use membership operator to check for the keys of dictionary
- `DI=dict(a=10,b=20,c=30)`

`Bool1='a' in DI`

`Bool2 = 'b' in DI`

`Bool3 = 10 in DI → view the output`

18 IDENTITY

- Using is operator

S1=S2

Res=S1 is S2

print(Res)

19 EMPTINESS

- Can be checked using not operator or bool function
- `D1=dict(a=10,b=20,c=30)`
- `bool(D1)`

`D2={}`

`print(bool(D2))`

Using 'not' operator

if not `D1`:

`print("Empty Dictionary")`

20 FUNCTIONS FOR DICTIONARY

- `len()`
- `max()`
- `min()`
- `sum()`
- `any()`
- `all()`
- `del()` – Here `del()` can be used to delete a dictionary or delete an item corresponding to a key
- `sorted()` – It will generate the sorted list of keys

21 METHODS FOR DICTIONARY

- `copy()` – deep copy
- `clear()` – to clear the contents of a dictionary (to create an empty dictionary)
- `update()` – to update an existing dictionary with the contents of another dictionary object
- The argument to be passed in `update()` should be of type dictionary
- `popitem()`- to remove the items in the dictionary following LIFO rule.
- `pop(key)` – to remove an item specified by the key

22 DICTIONARY COMPREHENSION

- Similar to that of list comprehension but has to comply with the validity of dictionary
- Case 1: Single for loop
- `{n: n*n for n in range(1,10)}`
- Try `{(n,n+1):n*n for n in range(1,10)}`
- Try `{[n,n+1]:n*n for n in range(1,10)}`
- Try `{ {n,n+1}:n*n for n in range(1,10)}`

23 DICTIONARY COMPREHENSION

- Case 2: for loop with if
- This will help to generate a dictionary for a given range when the condition given in if is True
- Example – generate a dictionary of keys that are divisible by 6 within the range of 0 to 100 and associate them with their squares
- $\{n:n*n \text{ for } n \text{ in range}(0,100) \text{ if } n\%6==0\}$

24 DICTIONARY COMPREHENSION

- Case 3 :for loop with if and else
- This will help to generate a dictionary taking two different expressions for a given range depending on the condition given in if clause turns to be True or False
- Example – generate a dictionary that takes first n natural numbers as the key and takes values $n*n*n$ when $n < 5$ and $n*n$ when $n > 5$ for a given range of (1,10)
- $\{n: (n*n*n \text{ if } n < 5 \text{ else } n*n) \text{ for } n \text{ in range}(1,10)$

25 SUMMARY OF CONTAINER TYPES

Characteristics	LIST	SET	TUPLE	DICTIONARY
Iterability	YES	YES	YES	YES
Indexability	YES	NO	YES	KEY INDEXING
Mutability	YES	YES	NO	YES
Duplication of items	YES	NO	YES	No duplication of keys
Ordered collection	YES	NO	YES	YES but not position indexable
Slicing	YES	NO	YES	NO
Comprehension	YES	YES	NO	YES

26 SUMMARY OF CONTAINER TYPES

Operation	LIST	SET	TUPLE	DICTIONARY
Concatenation	YES	NO	YES	YES with **
Aliasing	YES	YES	YES	YES
Cloning	YES	YES but with copy method only	YES	YES with ** or using copy()
Comparison	YES (item by item)	YES but superset or subset	YES (item by item)	NO
Searching	YES, using in operator	YES, using in operator	YES, using in operator	YES, but for keys only
Deletion of items	YES	Using built in methods only	NO	YES using keys
Nesting	YES	NO	YES	YES

Thank You





CS 200I PYTHON PROGRAMMING

2 LECTURE OBJECTIVES

- Introduce the concept of user defined functions in python

3 FUNCTIONS

- Self contained block of program statements intended to implement a specific task.
- Since C is a functional programming language, it will always have at least one function.
- In python, execution takes place line by line and hence there is no requirement of any function for small scripts.
- For better maintenance, debugging and for reuse of the codes with more number of tasks, python supports the implementation of user defined functions.
- Python provides more flexibility compared to C in terms of the usage of user defined functions

4 REQUIREMENTS OF FUNCTION

- To implement a user defined function in python, we need to have
 - Definition of the function which can be done before or after the main part of the code
 - Function call that can be done within the main part of the code.
- Syntax
 - Beginning of the user defined function is indicated with the keyword 'def'
 - `def<space>function name(argument names):`
 function body
 - Function call takes the form `<function name>(arguments)`

5 RETURN TYPE

- In python, user defined functions can be classified into
 - Void functions – that does not return any data
 - Fruitful function – that returns at least one data or object
- Unlike C, python has the provision to return multiple objects to the main code
- Function can return primary data objects and secondary data objects
- List, tuple, set or dictionary can be returned using the keyword 'return'

6 EXAMPLE I

- Void function

```
def display_data(p,q,r):
```

```
    print(p**2)
```

```
    print(round(q,2))
```

```
    print(r.title())
```

```
display_data(10,3.142876,"i love you")
```

7 EXAMPLE2

- Fruitful function

```
def sum_cubes(n1,n2,n3):
```

```
    sum3=n1**3+n2**3+n3**3
```

```
    return sum3
```

```
S=sum_cubes(1,2,3)
```

```
print("The result is",S)
```


8 ARGUMENT TYPES IN PYTHON

- Python supports the following types of arguments to be passed into a function

- Positional arguments

- Keyword arguments

} Required arguments

- Variable length positional arguments

- Variable length keyword arguments

} Optional arguments

9 POSITIONAL ARGUMENTS

- Python matches between the arguments in the definition and that in the function call based on their position.
- If there is a mismatch in the type due to change in position, an error will be occurred.
- Here, the arguments are passed by value and hence matching of type is essential to perform the task specified in the function definition.

10 EXAMPLE

Consider the example I given slide 6

Here the arguments are matched using the position.

If we give the function call as `display_data("i love you", 10, 3.142876)`, python will raise an error since it can't do

squaring of a string,

rounding an integer

string operation on a float

The first error will be displayed by python interpreter

II KEYWORD ARGUMENTS

- Here the arguments are matched based on the keyword
- Position of the arguments in the definition and call are irrelevant.
- The keywords in the definition and call should be identical.
- If the keywords do not match, python will raise an error

12 EXAMPLE

- We can modify the previous function as

```
def display_data(number, pi_n, cringe):
```

```
    print(number**2)
```

```
    print(cringe.title())
```

```
    print(round(pi_n,2))
```

```
display_data(cringe="i love you", number=10, pi_n=3.142876)
```

Here the values corresponding to the keyword will be passed into the function definition

I3 VARIABLE POSITIONAL ARGUMENTS

- Python provides the flexibility in changing the number of positional arguments when the function is called multiple times within the main code.
- In the function definition, variable positional argument is prefixed with '*'.
- It can be treated as unpacking a tuple.
- Within the function body, the argument can be iterated to get each value that are passed in the function call.

14 EXAMPLE I

- Example 2 in slide 7 can be modified as

```
def sum_cubes(*var_arg):
```

```
    sum3=0
```

```
    for j in var_arg:
```

```
        sum3+=j**3
```

```
    return sum3
```

```
S1=sum_cubes(1,2)
```

```
print("The first result is",S1)
```

```
S2=sum_cubes(1,2,3,4)
```

```
print("the second result is",S2)
```

15 EXAMPLE2

- To find the integer quotient and remainder of two numbers

```
def div_mod(*var_arg):
```

```
    q=var_arg[0]//var_arg[1]
```

```
    r=var_arg[0]%var_arg[1]
```

```
    return q,r
```

```
N1=int(input("enter the dividend\n"))
```

```
N2=int(input("Enter the divisor\n"))
```

```
T=div_mod(N1,N2)
```

```
print("The quotient and remainder are",T)
```

16 VARIABLE KEYWORD ARGUMENTS

- In variable positional arguments, each argument in the function call is taken in order in the function definition.
- A variable keyword argument is denoted in the function definition by prefixing '**' before the argument name.
- It can be considered as a variable length dictionary
- Variable keyword arguments enhances the flexibility of a user defined function by two fold
 - Provision for changing the number of arguments at the time of function call
 - Provision for accessing the values given in the function call by tracking the keyword associated with them instead of position.

17 EXAMPLE I

- Modifying the div_mod() function

```
def div_mod(**kw_args):
```

```
    q=kw_args['dividend']//kw_args['divisor']
```

```
    r= kw_args['dividend']%kw_args['divisor']
```

```
    return q,r
```

```
N1=int(input("enter the dividend\n"))
```

```
N2=int(input("enter the divisor\n"))
```

```
T=div_mod(divisor=N2,dividend=N1)
```

```
print("The quotient and remainder are\n",T)
```

18 EXAMPLE2

```
def Display_Data(**dI):
```

```
    for k in dI:
```

```
        print(k,dI[k])
```

```
Display_Data(cringe="I Love You")
```

```
Display_Data(number=10,cringe="I Love You")
```

```
Display_Data(pi_n=3.142876,number=10,cringe="I Love You")
```

19 TIPS

- Within a function definition, we can't have multiple variable positional arguments or multiple variable keyword arguments.
- There can be multiple positional arguments and multiple keyword arguments.
- Within a function, if all four types of arguments are to be included, they must take the following order in function definition and function call.
 - Positional arguments
 - Variable positional arguments,
 - Keyword arguments
 - Variable keyword arguments

20

Thank You





CS 200I PYTHON PROGRAMMING



2 LECTURE OBJECTIVES

- Introduce the concept of functional programming in python

3 FUNCTIONAL PROGRAMMING

- In functional programming
 - Problem is treated as the evaluation of one or more functions
 - If the problem is simple, it can be defined as a single function
 - If the problem is complex, it is decomposed into many functions
 - Python facilitates functional programming by treating functions as 'FIRST CLASS VALUES'

4 FIRST CLASS VALUES

According to this concept,

- a function can be assigned to a variable and whenever the function is required, the variable can be called
- A function can be passed as argument of another function and a function can be returned as the result of a function
- Functions can be built at the time of execution

5 FUNCTION AS A VARIABLE

```
def display_data(p,q,r):
```

```
    print(p**2)
```

```
    print(round(q,2))
```

```
    print(r.title())
```

D=display_data → assigning the function to a variable

D(10,3.1428,“god bless”) → calling the function by calling the variable

6 EXAMPLE2

- Fruitful function

```
def sum_cubes(n1,n2,n3):
```

```
    sum3=n1**3+n2**3+n3**3
```

```
    return sum3
```

```
S=sum_cubes
```

```
print("The result is",S(1,2,3))
```

7 PASSING FUNCTION AS ARGUMENT

```
def display_data(p,q,r,sum_cubes):  
    print(p**2)  
    print(round(q,2))  
    print(r.title())  
    print(sum_cubes(1,2,3))  
display_data(10,3.1428,'god bless',sum_cubes)
```

8 EXAMPLE 2

```
def sum_cubes(*var_arg):  
    sum3=0  
    for j in var_arg:  
        sum3+=j**3  
    return sum3  
  
S=sum_cubes
```

9 EXAMPLE 2 – CONT..

```
def display_data(p,q,r,s):  
    print(p**2)  
    print(round(q,2))  
    print(r.title())  
    print(s(1,2,3,4))
```

```
D=display_data
```

```
D(10,3.1428,'god bless',S)
```

10 FUNCTION BUILT AT THE EXECUTION TIME

Python supports defining a function at the point of execution

This is implemented with the help of special functions

They are not built using the keyword 'def'

Special functions do not possess names but are identified by their own keyword

'lambda' , 'map' and 'filter' are a few special functions provided by python.

II LAMBDA FUNCTIONS

- They are also called inline functions or anonymous functions
- They are built using the keyword 'lambda'
- Syntax is
lambda<space> arguments:expression
- Lambda function can take any number of arguments but can return only one value
- Arguments are separated by commas
- Return value corresponds to the expression

12 EXAMPLE I

To replace the function

```
def sum_cubes(n1,n2,n3):  
    sum3=n1**3+n2**3+n3**3  
    return sum3
```

We can define a lambda function as

```
lambda n1,n2,n3:n1**3+n2**3+n3**3
```

The definition and call can be at the same place as

```
(lambda n1,n2,n3:n1**3+n2**3+n3**3)(1,2,3)
```

I3 EXAMPLE 2

- Lambda can be used along with built in functions
- Consider the program of sorting a list of tuples
- `L1=["Titiksha","Samridhi","Deeksha","Shakti"]`
- `L2=[1060,2057,1082,1041]`
- `L3=[90,87,80,82]`
- `LT=list(zip(L1,L2,L3))`
- `LT'=sorted(LT, key= lambda x:x[2])`

14 LIMITATION OF LAMBDA

- Consider the example

```
def sum_cubes(*var_arg):
```

```
    sum3=0
```

```
    for j in var_arg:
```

```
        sum3+=j**3
```

```
    return sum3
```

```
print(sum_cubes(1,2))
```

```
print(sum_cubes(1,2,3,4))
```

It is difficult to replace this function with lambda

15 LIMITATION OF LAMBDA FUNCTION

- One possible solution for the previous problem is
- `(lambda *vargs:sum([j**3 for j in vargs]))(1,2,3)`
- Since it makes the expression complex, lambda function is normally used as replacement of simple user defined functions without if else, for, while etc

16 APPLICATION OF LAMBDA

- lambda functions are very useful when
 - It is required to define a simple expression as a function
 - It is required to use that expression only once
 - It is required to use different simple expressions at different parts of the code
- To process any iterable and to get a new iterable, `map()` is used

17 MAP FUNCTION

- map function helps to process the iterables without loop
- The syntax is
- `map(function, iterable)`
- Here function denotes the transformation function that is applied to an iterable to produce a new iterable.
- Items in the new iterable are produced by calling the transformation function on each item in the original iterable
- Like `zip()` function, `map()` also creates an object which can be accessed by the system only.
- We can convert the mapped object as a list , tuple etc by using converting functions

18 EXAMPLE I

- Exam question: reversing all numbers present in a list

```
def rev_number(number):
```

```
    r_number=0
```

```
    while(number!=0):
```

```
        r=number%10
```

```
        r_number=r_number*10+r
```

```
        number=number//10
```

- ```
 return r_number
```
- ```
LI=[int(n) for n in input("enter the numbers\n").split()]
```
- ```
Lrev=list(map(rev_number,LI))
```

## 19 EXAMPLE 2

---

- `map()` can be applied to built in functions in python
- For example, in the previous slide
- `L1=[int(n) for n in input("enter the numbers\n").split()]` can be modified as
- `L1=list(map(int,input("enter the numbers\n").split()))`

## 20 MAP() AND LAMBDA()

---

- `map()` can be applied to lambda function
- `T1 = tuple(map(int, input("enter the numbers\n").split()))`
- `T2 = tuple(map(lambda x: x**2, T1))`
- `L1 = input("enter the strings\n").split()`
- `L2 = list(map(lambda x: x.upper(), L1))`

## 21 FILTER FUNCTION

---

The function `filter()` is also similar to `map()`

It works on an iterable and will return an iterable

The filtering on the iterable will depend on the transformation function

The transformation function returns `TRUE` or `FALSE`

After filtering, the new iterable will consist of elements that will return `TRUE`

In both `map()` and `filter()`, the transformation function can be built in or user defined

## 22 EXAMPLE I

---

- Consider the case of checking for prime numbers present in a list

- `def is_prime(n):`

```
 h=n//2
```

```
 while(h>1):
```

```
 if(n%h==0):
```

```
 return False
```

```
 h=h-1
```

```
 else:
```

```
 return True
```

```
L1=[int(n) for n in input("Enter the numbers\n").split()]
```

```
L2=list(filter(is_prime,L1))
```

## 23 EXAMPLE2

---

- Filter can be a replacement for repetition control statement together with if clause
- For example
- `SI=input("enter the string")`

`L=list(filter(str.isalpha,SI))`



## 24 FILTER AND LAMBDA

---

- The transformation function can be defined using lambda function
- To generate a list of even numbers from a given list

```
L1=[1,2,3,4,5,6]
```

```
Leven=list(filter(lambda x:x%2==0,L1))
```

```
print(Leven)
```

- To generate a list of names starting with 'A' from the given name list

```
L=input("enter names\n").split()
```

```
L2=list(filter(lambda x:x.startswith("A"),L))
```

```
print(L2)
```

25

---

**Thank You**





---

# CS 200I PYTHON PROGRAMMING

## 2 LECTURE OBJECTIVES

---

- Introduce the concept of modules in python

# 3 FUNCTIONS

---

- Self contained block
- Enhances the readability of the code
- Debugging of the code
- Maintenance of the code
- Reusability of the code

## 4 MODULE

---

In general, any file with .py extension is treated as a module

If the file is run as standalone script, it is treated as 'main' module.

When the variables, classes and functions within a file are required in another python script, the file can be imported into that script using the syntax

```
import<space>filename
```

Normally, a collection of general purpose functions and classes are written a single python file and will be given a module name.

This module can be imported and can be used in any python script

Modules can be either built in or user defined

Modules as the name suggests, play key role in modular programming.



## 5 IMPORTING A MODULE

---

Consider a module with name `arithmetics.py`

This can be imported into another python script in the following ways

```
import arithmetics
```

```
import arithmetics as arit
```

The keyword 'as' helps to change the name of the module to 'arit' and thereby avoids any conflict with identifiers used in that particular script

The statement `import <module_name>` only places `<module_name>` in the caller's symbol table.

The objects that are defined in the module remain in the module's private symbol table.

## 6 SYMBOL TABLE

---

- Each python script carries a finite number of identifiers
- When the script is being interpreted, python generates a symbol table contains the details of all the identifiers related to that script
- The symbol table will comprise of the name given by the user, the type of the identifier, its scope and its location in the memory
- The names of identifiers available in the local symbol table can be find out using `dir()`

## 7 IMPORTING A MODULE

---

- If we import only the module name to the present script, to access the methods and variables of the module, we must use the syntax
- `modulename.method` or
- `modulename.variable`
- For example,

```
import math
```

```
math.sqrt(10)
```

## 8 IMPORTING OBJECTS AND FUNCTIONS

---

- To import all functions and objects to the present script, we need to use the syntax

`from<space>modulename<space>import<space>*`

To import a specific function, we can use the syntax

`from<space>modulename<space>import<space>function name` OR

`from<space>modulename<space>import<space>function name<space>as<space>new name`

Examples

`from math import *`

`from math import pow as p, sqrt as sq`

## 9 USER DEFINED MODULE

---

- To create a module, we have to open a file with .py extension
- Within the file, we can define as many functions or variables that are required.
- Function calls will not be a part of the module
- Normally, a module created for other python files may not run as standalone script.
- If the user intends to run the module as standalone script for certain functions, the calls can be made within the module by using the syntax

if (`__name__` is `__main__`):

    function call

## 10 EXERCISE

---

- Create a module with name `my_module` to incorporate the following functions.
- To find the squares of all numbers in a list, set or tuple
- To find the cubes of all numbers in a list, set or tuple
- To find the positive numbers in a list, set or tuple

(If the module is run as standalone script, display “ `my_module` is now available” )





**Thank You**





---

# CS 200I PYTHON PROGRAMMING

## 2 LECTURE OBJECTIVES

---

- Explore more about modules in python

### 3 EXERCISE

---

- Create a module with name `my_module` to incorporate the following functions.
- To find the sum of all numbers in a list, set or tuple
- To find the maximum of all numbers in a list, set or tuple
- To find the minimum numbers in a list, set or tuple

(If the module is run as standalone script, display “ `my_module` is now available” )

## 4 NAMESPACES

---

- A dictionary of identifiers with key as the name of the identifier and value as the data assigned to that identifier
- There can be global or local namespaces.
- Namespaces can be displayed onto the screen using the functions `globals()` and `locals()`.
- For the main part of the code, local and global namespace has the same meaning.
- Within a function local and global namespaces will have different meaning.

## 5 GLOBAL AND LOCAL

---

- All the identifiers used within a function are local to that function.
- The scope of these variables are confined to that function only.
- All the identifiers used in the main code are global
- Within a python file if a global identifier and local identifier are given with the same variable name, within the function, the local value will be taken.
- If the global value is to be taken, the variable name should be prefixed with keyword 'global'



## 6 EXAMPLE



```
IDLE Shell 3.11.4
File Edit Shell Debug Options Window Help

{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__': <class '_frozen_importlib.BuiltinImpor
ter'>, '__spec__': None, '__annotations__': {}, '__builtins__': <module 'builtins' (built-in)>, '__file__': 'C:\\Users\\
user\\AppData\\Local\\Programs\\Python\\Python311\\Demo4.py', 'L': [1, 2, 3, 4], 'T': (5, 6, 7, 8), 'A': 10, 'S': {1,
2, 3, 4, 5}, 'Name_Spcae': <function Name_Spcae at 0x00000299B537BBA0>}

{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__': <class '_frozen_importlib.BuiltinImpo
rter'>, '__spec__': None, '__annotations__': {}, '__builtins__': <module 'builtins' (built-in)>, '__file__': 'C:\\Users\\
user\\AppData\\Local\\Programs\\Python\\Python311\\Demo4.py', 'L': [1, 2, 3, 4], 'T': (5, 6, 7, 8), 'A': 10, 'S': {1,
2, 3, 4, 5}, 'Name_Spcae': <function Name_Spcae at 0x00000299B537BBA0>}

Within the function
{'L': [5, 6, 7, 8], 'T': (9, 9, 9, 9), 'S': {1}}

{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__': <class '_frozen_importlib.BuiltinImpo
rter'>, '__spec__': None, '__annotations__': {}, '__builtins__': <module 'builtins' (built-in)>, '__file__': 'C:\\Users\\
user\\AppData\\Local\\Programs\\Python\\Python311\\Demo4.py', 'L': [1, 2, 3, 4], 'T': (5, 6, 7, 8), 'A': 10, 'S': {1,
2, 3, 4, 5}, 'Name_Spcae': <function Name_Spcae at 0x00000299B537BBA0>}
```

## 7 SOME IMPORTANT MODULES IN PYTHON

---

- random module
- 'datetime' module
- 'time' module

## 8 RANDOM MODULE

---

- Mainly applied in the analysis of random process
- Various methods are available
  - `random()`
  - `randint()`
  - `randrange()`
  - `choice()`
  - `shuffle()`

## 9 RANDOM()

---

- `random()`
- Function without any arguments
- Returns a float value between 0 and 1

Example

```
import random
```

```
random.random()
```



## 10 RANDINT()

---

`randint(a, b)` – to generate a random integer variable

Return random integer in range `[a, b]`, including both end points.

Example

```
import random
a=random.randint(1,10)
print(a)
```

## II RANDRANGE()

---

- To generate a random number by considering three arguments
- Start, stop and step
- In randint() by default the step is always 1.
- randrange() eliminates this drawback
- Example
- `P=randrange(1,100,10)`
- `print(P)`



## 12 CHOICE

---

- To randomly choose a variable from a given array of variables ( string, list or tuple)
- Example

```
from random import *
```

```
A=choice([11,3,4,7,9])
```

```
B=choice((1.3,3.4,5.6,7.8))
```

```
C=choice("Ignited Minds")
```

## 13 SHUFFLE

---

- To shuffle a given list of data
- Here the input argument cannot be a set , string or tuple
- The method shuffles the list in place and returns None
- Example
- `LI=[12,19,24,35,45,57]`
- `random.shuffle(LI)`
- `print(LI)`

## 14 DATETIME MODULE

---

- To deal with dates and time of each date
- Has various classes and methods
- Important classes are datetime, date ,time and timedelta

## 15 DATETIME CLASS

---

- We can create instances of datetime class
- `from datetime import *`
- `DateTime1=datetime(year=2023,month=11,day=6, hour =10, minute=5, second =50)`
- `Print(DateTime1)`
- `DateTime2=datetime.now()`
- `Print(DateTime2)`
- `Date3=date.fromtimestamp(1000000)`
- `Print(Date3)`

## 16 DATE CLASS

---

- The instances of date class can be created by using the keyword date
- `Date1=date(year=2023,month=11,day=6)`
- `print(Date1)`
- `Date2=date.today()`
- `print(date2)`

## 17 TIME CLASS

---

- Each instance carries the details of certain time instant
- Time object is created using the keyword time
- `Time1=time(hour=22,minute=24,second=10,microsecond=123456)`
- `Print(Time1)`



## 18 STRFTIME FUNCTION

---

- It helps to return the string corresponding to the integer arguments present in the datetime object, date object or time object
- Integer to string conversion is done with the help of various format specifiers
- Example

```
Date3=date.fromtimestamp(1000000)
```

```
print(Date3)
```

```
Date33>Date3.strftime("%A,%B,%Y")
```

```
print(Date33)
```

## 19 FUNCTIONS OF TIME CLASS

---

| Function Name   | Description                                                       |
|-----------------|-------------------------------------------------------------------|
| dst()           | Returns tzinfo.dst() is tzinfo is not None                        |
| fromisoformat() | Returns a time object from the string representation of the time  |
| isoformat()     | Returns the string representation of time from the time object    |
| replace()       | Changes the value of the time object with the given parameter     |
| strftime()      | Returns a string representation of the time with the given format |
| tzname()        | Returns tzinfo.tzname() is tzinfo is not None                     |
| utcoffset()     | Returns tzinfo.utcoffsets() is tzinfo is not None                 |

## 20 FORMAT SPECIFIERS

- %d: Returns the day of the month, from 1 to 31.
- %m: Returns the month of the year, from 1 to 12.
- %Y: Returns the year in four-digit format (Year with century). like, 2021.
- %y: Returns year in two-digit format (year without century). like, 19, 20, 21
- %A: Returns the full name of the weekday. Like, Monday, Tuesday
- %a: Returns the short name of the weekday (First three character.). Like, Mon, Tue
- %B: Returns the full name of the month. Like, June, March
- %b: Returns the short name of the month (First three character.). Like, Mar, Jun
- %H: Returns the hour. from 01 to 23.
- %I: Returns the hour in 12-hours format. from 01 to 12.
- %M: Returns the minute, from 00 to 59.
- %S: Returns the second, from 00 to 59.

## 21 FORMAT SPECIFIERS

---

- %f: Return the microseconds from 000000 to 999999
- %p: Return time in AM/PM format
- %c: Returns a locale's appropriate date and time representation
- %x: Returns a locale's appropriate date representation
- %X: Returns a locale's appropriate time representation
- %z: Return the UTC offset in the form  $\pm$ HHMM[SS[.ffffff]] (empty string if the object is naive).
- %Z: Return the Time zone name (empty string if the object is naive).
- %j: Returns the day of the year from 01 to 366
- %w: Returns weekday as a decimal number, where 0 is Sunday and 6 is Saturday.
- %U: Returns the week number of the year (Sunday as the first day of the week) from 00 to 53
- %W: Returns the week number of the year (Monday as the first day of the week) from 00 to 53

**Thank You**





---

# CS 200I PYTHON PROGRAMMING



## 2 LECTURE OBJECTIVES

---

➤ Explore numpy

### 3 NUMPY

---

- Numerical Python
- One among the fundamental packages in python
- Deals with single and multidimensional numerical arrays
- Has variety of subpackages like random, fft, linalg etc
- Provides a number of classes and methods that makes numerical data processing handy

## 4 ARRAY

---

- One of the most important class in numpy
- Helps to convert any input into 'n' dimensional numerical array

Example:

```
import numpy as np
```

```
LI=[1,2,3,4]
```

```
AI=np.array(LI)
```

```
print(AI) → see what is displayed
```

```
print(type(AI)) → see what is displayed
```

## 5 ARRAY DATATYPES

---

- Array can take integer(signed or unsigned), float, complex, Boolean and string data.
- If the datatype is not mentioned, numpy assigns the default datatypes
- We can customize the datatype by adding the attribute 'dtype' along with array()
- `A=np.array(L1,dtype='i')`
- `print(A)`
- Integer datatype can be of 2 bytes, 4 bytes, 8 bytes etc. So dtype can be 'i2', 'i4' or 'i' , 'i8'
- Unsigned can be of 2 bytes, 4 bytes, 8 bytes etc. That means dtype can be 'u2', 'u4' or 'u8'
- Similarly, float can be 'f2', 'f4' or 'f8'
- Complex can be either 'c8' ,or 'c16' .

## 6 BOOLEAN

---

- For Boolean , simply give dtype='?' or dtype=bool

**Example:**

```
L=[0,1,2,3]
```

```
A=np.array(L, dtype='?')
```

```
print(A)
```

The output will be like

```
array[False,True,True,True]
```

## 7 STRING

---

- Even though NumPy is primarily used for numerical data, it supports 'strings' too
- In Numpy , strings are characterized by Unicode values
- So array of strings can be created by setting dtype='Un' or by passing a list of strings into the array class
- 'U' indicates Unicode and 'n' indicates the number of characters present in the string.

### **Example**

```
L=["Narendra", "Damodar", "Das", "Modi"]
```

```
A=array(L)
```

```
print(A)
```



## 8 ARANGE

---

- The function `arange()` helps to create an integer array object
- `A1=np.arange(1,10,2)`
- `print(A1)`
- The output will be `array([1,3,5,7,9])`
- `print(type(A1))`
- The output will be `<class numpy.ndarray>`
- Here by default datatype is 'int'
- `A1=np.arange(1,10,2,dtype='f') → try`

## 9 ZEROS, ONES AND IDENTITY MATRICES

---

- `A0=np.zeros((3,3),dtype='i8')`
- `A1=np.ones((3,3),dtype='f8')`
- `Aid=np.eye(3)`

## 10 BASIC INDEXING AND SLICING

---

- `A[0]` → returns the first row of the 2D array as a 1D array
- In the case of 1D array, `A[0]` will return the first element
- `A[-1]` → last row as 1D array of the 2D array
- In the case of 1D array, `A[-1]` will return the last element
- `A[:,0]` → returns the first column as 1D array
- `A[:, -1]` → last column
- `A[0,0]` → returns the first element as a scalar value for a 2D array
- `A[:,0:1]` → try

## II MUTABILITY

---

- Array is a mutable object.
- Modification can be done by subscripting or by built in methods/functions
- Example

```
L=[0,1,2,3]
```

```
A=np.array(L)
```

```
print(A)
```

```
A[0]=5
```

```
print(A)
```

## 12 BOOLEAN INDEXING

---

- Unlike container types, numpy provides Boolean indexing of elements
- `L=[1,4,7,11,15,20,32]`
- `A=np.array(L)`
- `A[A%4==0]` will return the array `[4,20,32]`,
- Conditional expressions can be combined using `logical_and` or by using `logical_or`
- `A[np.logical_or(A%4==0,A%5==0)]`

## I3 CONDITIONAL EXPRESSION USING 'WHERE'

---

- The function `where()` helps to modify the contents of specific locations of the array
- `A=np.array([1,2,3,4,5,6,7,8],dtype='i4')`
- `A=np.where(A%2==0,0,1)`
- `print(A)`



## 14 FANCY INDEXING

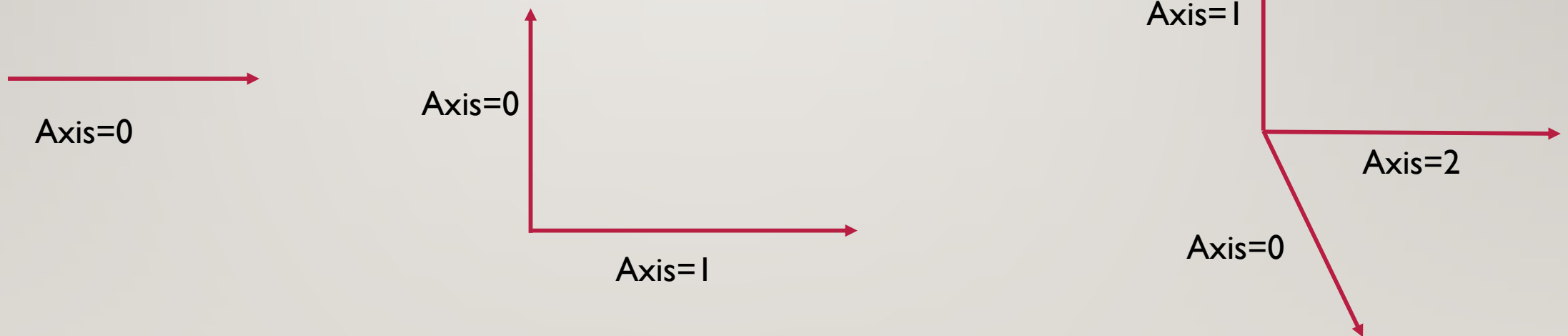
---

- This helps to give an easy indexing of arrays.
- The index can be a list , an array or a sequence of integers.
- Fancy indexing helps to return the correct dimensions of the array.

## 15 AXES AND THEIR NAMING CONVENTION

---

- For 1 D array, the axis taken as horizontal and is named as axis=0
- For 2 D array, there are two axes, horizontal is named as 1 and vertical as 0



## 16 ARRAY METHODS

---

- `Al.size` → returns the number of elements
- `Al.ndim` → returns the number of dimensions
- `Al.shape` → returns the length of the array along each axes as a tuple
- `Al.reshape` → to change the dimensions according to user's preference
- `Al.ravel` → to make the array into 1 dimensional (linear sequence)
- `Al.sum()` → find the sum of all elements( can be applied axis wise)
- `Al.sort()` → to sort the elements, by default sorts on axis 1 (sorting of individual rows)
- `Al.copy()` → to make a deep copy
- `Al.view()` → to make a shallow copy

## 17 ITERATING THE ELEMENTS

---

- 1D array

```
for j in A:
```

```
 print(j)
```

- 2D array

```
for j in A.shape[0]:
```

```
 for k in A.shape[1]:
```

```
 print(A[i][j])
```

Note: modify the nested for loop to access each element of 2D array without using shape method

## 18 IMPORTANT UNARY FUNCTIONS

---

- sum
- sqrt
- abs, ceil, floor
- modf
- sign
- trace
- sort

## 19 IMPORTANT BINARY FUNCTIONS

---

`np.add`

`np.subtract`

`np.matmul`

`np.divide`

`np.concatenate, np.vstack, np.hstack`



## 20 MATRIX OPERATIONS

---

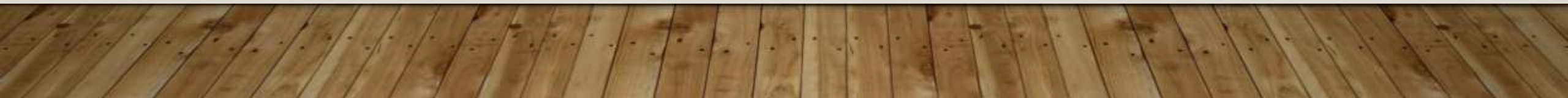
- To find the transpose of a matrix  $\rightarrow A.T$  will return the transpose of A
- To perform addition use '+' operator
- To perform subtraction use '-' operator
- To perform multiplication, use '@' operator
- To perform division , use '/' operator

## 21 DETERMINANT AND INVERSE

---

- The subpackage 'linalg' provides functions to find the determinant and inverse
- `np.linalg.det(A)` will return the determinant
- `np.linalg.inv(A)` will return the inverse

**Thank You**





---

# CS 200I PYTHON PROGRAMMING

## 2 LECTURE OBJECTIVES

---

- Outline the general instructions
- Introduce the course objectives
- Discuss the course syllabus and course plan
- Brief the concept of programming Languages

### 3 GENERAL INSTRUCTIONS

---

- No need to greet the teacher outside the class
- Be punctual for the class or lab
- Strictly, no use of mobile phones during class
- Keep the decorum of the class



## 4 LECTURE PATTERN

---

- 20 minutes – introduction and explanation of a concept
- 20 minutes – interactive and hands on session
- 10 minutes - student's choice

## 5 THEORY COURSE ASSESSMENT CRITERIA

---

OUT of 100 marks

- ❖ 50 for End Sem Exam, 30 for Mid Sem Exam
- ❖ 10 marks for group assignment
- ❖ 10 marks for attendance ( If total attendance is  $\geq 80\%$ )
- ❖ 3 bonus marks for participation in the student's choice session

## 6 COURSE SYLLABUS

---

|          |                                         |
|----------|-----------------------------------------|
| UNIT I   | : Basics of Python                      |
| UNIT II  | : Control Statements in Python          |
| UNIT III | : List, Tuple, Sets and Dictionary      |
| UNIT IV  | : Functions and Modules in Python       |
| UNIT V   | : File Management                       |
| UNIT VI  | : Matplotlib, numpy and pandas          |
| UNIT VII | : Machine Learning and IoT using Python |

## 7 COURSE PLAN – PHASE I

---

- Week 1 : Revision of C and Installation of Python
- Week 2 : Introduction to Python – Datatypes, operators and expressions
- Week 3 : Control Statements in Python
- Week 4 and 5 : List, tuple, set, dictionary
- Week 6 : Functions and Modules
- Week 7 : Review of first 6 weeks

## 8 COURSE PLAN – PHASE 2

---

- Week 8 : File management in Python
- Week 9 : Plotting data using matplotlib
- Week 10 : Basics of Numerical Python – Numpy
- Week 11: Panel data Analysis – Pandas
- Week 12 : Machine learning and Python
- Week 13 : IoT and Python
- Week 14 : Final Review

## 9 PROGRAMMING LANGUAGES

---

- Program - Medium to invoke a computer for the fulfilment of certain task
- Etymology – Greek words ‘pro’ ( before) and ‘graphein’(to write)
- Various types of programming languages are available at present
- Based on its closeness to hardware, classified as LOW, MEDIUM and HIGH
- Based on code orientation, classified as
  - Functional Programming
  - Procedural Programming
  - Object Oriented
  - Even Driven



# 10 STEPS TO LEARN PROGRAMMING

---

- Learn the alphabet - Keyboard of the programming language
- Learn to use words – Keywords , identifiers, operators
- Learn to make a sentence – Use keywords, identifiers, operators etc to make a meaningful program statement
- Learn to prepare a paragraph – A block of code (decision control, loop control, function)
- Learn to prepare an article – Write a complete program
- Learn to write a book – Write codes for specific applications

# II SCRIPT WRITING AND CODING

---

- Writing code/ program is analogous to script writing
- An eye on details
- An eye on continuity

## I2 REVISITING C

---

- Character Set
- Keywords and identifiers
- Datatypes
- Operators
- Control Statements
- Function and recursion
- Structures

# I3 CHARACTER SET IN C

---

Alphabet of C

To write programs in C, we can use

English alphabet – both uppercase and lowercase

Digits – 0 to 9

Operators – Arithmetic, Logical, Comparison, bitwise

Special Symbols – From ~ to \_

comma, column, single quote, double quote

parenthesis, braces, square brackets

## 14 KEYWORDS AND IDENTIFIERS

---

- Keywords – Reserved set of words that are provided by the C compiler having predefined meaning and specific purpose

Keywords are case sensitive and are always in lowercase

Eg : int, float, break, for, switch

Identifiers – names given by the programmer for variables, functions, structures etc

Identifiers shall adhere to the rules provided by C compiler

# 15 DATATYPES IN C

---

- Static datatype assignment in C
- To be assigned at the time of variable declaration or initialization
- Datatype defines the memory requirement of the variable
- Datatype can be
  - Primary – int, float, char
  - Secondary or derived – array , matrix, string
  - User defined



# 16 OPERATORS

---

- Arithmetic operators
- Logical Operators
- Conditional operators
- Bitwise Operators
- Assignment operator

# 17 CONTROL STATEMENTS IN C

---


- Sequential Control Statements
- Decision Control Statements
- Loop Control Statements
- Case Control Statements

## 18 DECISION CONTROL STATEMENTS

- Different formats

---

|               |                 |                    |
|---------------|-----------------|--------------------|
| if(condition) | if ( condition) | if(condition)      |
| {             | {               | {                  |
| Do this ;     | do this;        | do this;           |
| }             | }               | }                  |
|               | else            | else if(condition) |
|               | {               | {                  |
|               | do this;        | do this;           |
|               | }               | }                  |
|               |                 | else               |
|               |                 | {                  |
|               |                 | do this;           |
|               |                 | }                  |



## 19 LOOP CONTROL STATEMENTS

---

- while(TRUE)

```
{
 do this;
}
```

- do

```
{
 do this;
}
while (TRUE);
```

- for(i=0;i<n;i++)

```
{
 do this;
}
```

## 20 CASE CONTROL STATEMENTS

---

```
switch(choice)
```

```
{
```

```
case 1:
```

```
{
```

```
do;
```

```
break;
```

```
}
```

```
case 2:
```

```
{
```

```
do;
```

```
break:
```

```
}
```

```
}
```

## 21 FUNCTIONS

---

- C is an example of functional programming
- Each C program will have at least one function
- By default function is main()
- Execution of C program begins from main() function
- There can be n number of user defined functions
- Function is a self contained block of program statements to fulfil a certain task
- In C, each function should have declaration, definition and function call



## 22 RECURSION

---

- Normally a function is called within main()
- When a function is called within its definition it is called recursion
- Recursion can create infinite loop if not set a proper condition

## 23 STRUCTURE IN C

---

- User defined datatype to accommodate multiple datatypes
- Each variable within a structure is called structure member
- Struct variable can access all these members using 'dot' operator
- Struct enables to write more sophisticated programs having data manipulation.

## 24 DRAWBACK OF STRUCTURE

---

- We can't define a function within a structure
- It does not support object oriented programming (OOP)
- Most of the real world problems can be easily solved with the help of OOP

## 25 REVISION EXERCISES

---

1. Write a program to find the equivalent resistance of two resistances connected in parallel. Enter the value of R1 and R2 at run time.
2. Write a program to check whether the given number is prime or not
3. Write a program to sort the numbers of a given array
4. Rewrite the previous program (Q. 4) using functions
5. Write a program to find the factorial of a number.
  - by using function.
  - without using function

**Thank You**





---

# CS 200I PYTHON PROGRAMMING



## 2 LECTURE OBJECTIVES

---

- Introduce the concept of Object Oriented Programming
- Brief the salient features of Python
- Provide guidance regarding the installation of python

### 3 OBJECT ORIENTED PROGRAMMING

---

- A programming concept that was developed in the modern era.
- A different perspective towards real world problems that include large amount of data
- The concept of structure was modified and a new datatype called class was defined.
- Class can be user defined or can be built in, similar to that of function.
- Within a class, there can be finite number of dissimilar datatypes
- Classes will have the provision to define their own functions (methods)
- Variables of a class are called instances of that class or objects of that class
- There can be constructor functions within which the instance of the class can be passed as argument

## 4 EXAMPLE FOR A CLASS

---

```
class train
{
 // characteristics
 int train_no;
 char destination;
 char train_type;
 int arrival_time;
 int departure_time;
```

```
 int arrival(delayed_time)
 {
 arrival_time += delayed_time;
 return arr_time;
 }

 int departure(delayed_time)
 {
 departure_time += delayed_time;
 return departure_time;
 }
}
```

# 5 FEATURES OF OOPS

---

- ❖ Data Encapsulation
- ❖ Data abstraction
- ❖ Polymorphism
- ❖ Inheritance

## 6 DATA ENCAPSULATION

---

- In object oriented programming we can define 'class' to bind related attributes and functions into a single unit
- All members of the class can be accessed outside the class by creating an object of that class
- Binding dissimilar datatypes and functions into a single unit is called encapsulation



## 7 DATA ABSTRACTION

---

- Data abstraction comes along with encapsulation.
- It can be called as data hiding also
- Functions or variables within a class can be hidden from other part of the program.
- In C++, data abstraction is done with the help of access priority
- Three types of access specifiers are used in C++- public, private and protected.
- Python supports abstract class and abstract methods



## 8 INHERITANCE

---

- The literal meaning of this word is adopted to describe the property of classes
- In OOPs, any class can have its derived class or subclass.
- Derived class can inherit the features of the parent class.
- For Example, there can be a parent class 'Employee' within which we can define derived classes like "Manager", "Accountant", "Front officer" etc.
- Inheritance enables to have better codes with fewer lines.

## 9 POLYMORPHISM

---

- Polymorphism implies same name- different tasks
- C++ supports functional polymorphism and operator overloading
- Functional polymorphism – Within a single program we can have many functions with the same name but difference in the type and number of arguments
  - `void Area( int, int):`
  - `void Area(float, float);`
  - `void Area(int, int, int);`
- Operator overloading refers to the multiple use of the same operator.

# PYTHON

---

- High level, object oriented, programming Language
- Developed by Dutch scientist Guido Van Rossum in 1991
- Named after a popular TV show 'Monty Python's Flying Circus'
- One of the most popular programming language of 21<sup>st</sup> century
- Scripting Language ( Line by line execution)

# II FEATURES OF PYTHON

---

- Open Source
- Support of packages or libraries
- Software quality
- Developer productivity
- Program portability
- Component integration

## I2 DOMAINS OF APPLICATION

---

Scientific Programming

- NASA, Fermi Lab

Movie Animation

- Pixar, Industrial light and magic

Financial market forecasting

- JP Morgan

Hardware Testing

- Qualcomm, IBM, HP

Video Sharing

- YouTube

.....many more

# I3 WORKING WITH PYTHON

---

- Scripting mode:
  - Write a complete program in an editor, save it and run
  - Not for beginners
- Interactive mode:
  - Write the code line by line and execute in a python shell
  - Good for beginners



## I4 INSTALLATION OF PYTHON

---

- Go to [www.python.org](http://www.python.org)
- Click on downloads
- Choose the appropriate version
- Download it and run the .exe file
- tick 'Add to the path' dialog box and install
- Go to the command window and type `python --version`
- If the installation is complete, the version of python will be displayed

## 15 INSTALLATION OF JUPYTER

---

- Go to windows power shell ( command window)
- Type pip install jupyter
- Create a folder in your preferred location
- Open the folder, right click, select open in terminal
- Terminal with folder name as directory will appear
- Type jupyter notebook
- Jupyter notebook tab will be opened

- 
- <https://realpython.com/jupyter-notebook-introduction/>



---

# CS 200I PYTHON PROGRAMMING

## 2 LECTURE OBJECTIVES

---

- Discuss python basics
  - Character set
  - Keywords and identifiers
  - Basic datatypes
  - Built in functions useful for basic datatypes
  - Console input -output

### 3 CHARACTER SET IN PYTHON

---

Similar to that of C

English alphabet – both uppercase and lowercase

Digits – 0 to 9

Operators – Arithmetic, Logical, Comparison, bitwise

Special Symbols – From ~ to \_

comma, column, single quote, double quote

parenthesis, braces, square brackets



## 4 KEYWORDS

---

- Keywords – Reserved set of words that are provided by the python interpreter having predefined meaning and specific purpose

Keywords are case sensitive and are always in lowercase

Eg : int, float, break, for, in

To see all keywords in python use the following commands

```
import keyword
print(keyword.kwlist)
```

## 5 IDENTIFIERS

---

- Variable names
- names given by user/programmer
- Rules to remember
  - Keywords should never be used as identifier
  - Can start with either letter(lowercase or uppercase) or underscore only
  - Can have any letter in the alphabet, digits, and underscore
  - Shall not use operators or special symbols
  - Eg : Num, num, NUm2, dataset, Data\_Set, \_Data are valid identifiers

## 6 DATATYPES IN PYTHON

---

- Dynamic datatype assignment in Python
- Type of the variable is determined at run time
- High flexibility for variable initialization
- Datatype defines the memory requirement of the variable
- Datatype can be
  - Primary – int, float, complex, bool, str
  - Secondary or container type – list, dictionary, tuple, set
  - User defined -classes

## 7 PRIMARY DATATYPES

---

- In python all basic datatypes are built in classes defined by python
- All basic datatype variables are objects of that class
- Typing `a=5` creates an object of type class 'int', typing `a=5.25` creates an object of type class 'float'
- Type of a variable can be determined by using the function `type()`
- Arbitrary precision of integers - Rather than storing values as a fixed number of bits related to the size of the [processor register](#), these implementations typically use variable-length arrays of digits.
- Floating point objects - Float values in Python are represented as 64-bit double-precision values.  $1.8 \times 10^{308}$  is the maximum value for any floating-point number. If the input value exceeds this max value, the Python program will return an error

## 8 INSTANTIATION

---

- `a=6`
- Creating an object of any class is called instantiation. ( creating an instance of a class)
- We can do multiple instantiations by a single line of code
- `a, b=6,7`
- `a=6;b=7`
- `a =b=7`
- `a,b=6,7.35`

## 9 TIPS

---

- Integer data should not start with zeros which will lead to syntax error

Eg `A= 00125` is invalid in Python

- Integer delimiter is not comma but underscore
  - Eg `A =1,23,456` is invalid
  - `A =1_23_456` is valid and will be treated as `123456`



## 10 BINARY, OCTAL AND HEXADECIMAL

---

- `x = 0b11011000`

- `print(x)`

- `x = 0b_1101_1000`

- `print(x)`

- `print(type(x))`

- `y = 0o123`

- `print(y)`

- `y = 0o_123`

- `print(y)`

- `print(type(y))`

- `z = 0x19A`

- `print(z)`

- `z = 0x_19A`

- `print(z)`

- `print(type(z))`

## II OBJECT AS VARIABLES

---

- Objects are nameless and their address is stored in the variable.
- Address of the object can be accessed using the function `id()`
  - `A=30` will create a nameless object whose address is stored in A
- If two variables are assigned with same constant value, instead of two objects, single object will be created by python.
- Objects can be passed as arguments in a function.

A

14770890034

30

14770890034

## 12 COMPLEX NUMBERS

---

- Python supports wide variety of data types unlike C
- To perform complex number computing, python has a built in class 'complex'
- Typing  $A=4+5j$  , creates an object A of type complex
- We can generate the number  $4+5j$  by typing `complex(4,5)`
- There are a few built in functions that can operate on complex variables

## 13 BOOL

---

- True and False data can be stored as bool objects.
- Arithmetic and logical operations are possible on bool objects
- All non zero numbers are treated as True
- Zero is treated as False

## 14 TYPE CONVERSIONS

---

- By using built in functions
- `int(float)` will yield the integer part of a floating point number
- `complex(int)` will yield a complex number with imaginary part 0

## 15 MATH FUNCTIONS

---

- Python provides numerous built in math functions to operate on basic datatypes
- List of math functions can be displayed onto the screen by typing

```
import math
dir(math)
```
- Required values can be obtained by passing object as argument to these functions
- Eg `abs(x)`, `pow(x,y)`, `max(a,b,x)`, `round(x,n)`



# 16 STRINGS

---

- Unlike C, in python, the fundamental non numerical datatype is string
- Python has a built in class 'str'
- Text data can be stored to a string object in different ways
- `name ='Manju Mathew'`
- `name ="Manju Mathew"`
- `name =""Manju Mathew""`
- `Name= input("Enter your name")`

# 17 PROPERTIES OF STRINGS

---

- String is an ordered collection –
  - Elements are stored in the order in which they are inserted
  - Elements can be accessed using an index
- Strings are iterable –
  - string can be iterated within a loop, each member of the string has its own index
- Strings are immutable –
  - Once created, a string cannot be modified by changing any element within the string.
  - We can create a new string with the same variable name.

## 18 INDEXING OF STRING

---

|    |    |    |    |    |    |
|----|----|----|----|----|----|
| 0  | 1  | 2  | 3  | 4  | 5  |
| P  | Y  | T  | H  | O  | N  |
| -6 | -5 | -4 | -3 | -2 | -1 |

String indexing can be positive or negative

Positive indexing starts from the first letter of the string and begins with 0 index

Negative indexing starts from the last letter of the string and begins with -1 index

# 19 OPERATIONS ON STRINGS

---

- Forming substrings – general format is `string[start:stop:step]`
  - `String1 = "Shakespeare"`
  - `sub1 = String1[0:5]` – if step is not mentioned, by default step is 1
  - `Sub2 = String1[0:6:2]`
  - `sub3 = String1[:1]` – start at 0 end at the last letter with step 1 – replication of the string
  - `Sub4 = String1[: -1]` – try it
- Concatenation of strings
  - `S1 = "Star"`
  - `S2 = "dom"`
  - `S3 = S1 + S2`
  - `S4 = S1 * 2`

## 20 BUILT IN FUNCTIONS

---

- `dir(str)` will display the list

## 21 CONSOLE INPUT OUTPUT

---

Display onto the screen

- We can display any variable, text or value using print() function
  - `print("Python Programming")`
  - `print(45678)`
  - `A=10`
  - `print(A)`
  - `print("I guess 'python' is not that tough")`



## 22 DATA INPUT

---

- Python 3 version by default supports entry of string through keyboard

```
Data=input("Enter the number\n")
```

```
12
```

```
print(Data)
```

```
Data=int(input("enter the number\n"))
```

```
12
```

```
print(Data)
```



---

# CS 200I PYTHON PROGRAMMING

## 2 LECTURE OBJECTIVES

---

- Discuss about the operators in Python

# 3 OPERATORS

---

- Arithmetic operators
- Logical Operators
- Bitwise Operators
- Comparison operators
- Assignment operator
- Short hand Assignment operators

## 4 ARITHMETIC OPERATORS

---

- Addition +
- Subtraction -
- Multiplication \*
- Fractional division /
- Integer division //
- Modulus %
- Exponentiation \*\*
  - $4^{**}3$  gives  $4*4*4$

## 5 MORE ABOUT ‘//’ AND ‘%’

---

- `//` is the floor division operator
  - It returns the smaller value less than or equal to the quotient
  - `7//2` gives `+3` ( actual quotient 3.5)
  - `-7//2` gives `-4` (actual quotient -3.5)
- `%` is the modulus or remainder operator
  - Operation `a%b` returns `a-(b*(a//b))`
  - `7%2` returns
  - `-7%2` gives



## 6 PRECEDENCE

---

- Priority of operators

()

\*\*

\*,/,//,%

+,-

## 7 ASSOCIATIVITY

---

- When there is a tie between operators, associativity plays the role
- Associativity can be either **left to right** or **right to left**
  - $A * B / C$  will do  $A * B$  first and then  $/$  (Left to right)
  - $A ** B ** C$  will do  $B ** C$  first and then  $**A$  (Right to left)
  - $A+B-C$  will do  $A+B$  first (left to right)

## 8 LOGICAL OPERATORS

---

- Unlike C, python uses text to denote logical operations
- 'and' for logical AND
- 'or' for logical OR
- 'not' for Logical NOT

Logical operations will yield Boolean value as output (True or False)

## 9 BITWISE OPERATORS

---

- $\sim$  for NOT
- $\gg$  and  $\ll$  for bitwise right and left
- $\&$  for AND
- $|$  for OR
- $\wedge$  for bitwise Ex-OR

# 10 COMPARISON OPERATORS

---

- >
- >=
- <
- <=
- ==
- !=

Comparison operations will yield Boolean variable as output

## II ASSIGNMENT OPERATOR

---

- '=' is called assignment operator
- Similar to C, assignment is from right to left

A=5

B=10

A=B

This statement will copy the content of B to A

print(A)



## I2 SHORT HAND OPERATORS

---

- $A+=5$  implies  $A = A+5$
- $A*=10$  implies  $A = A *10$

## I3 MEMBERSHIP OPERATORS

---

- 'in' and 'not in'

The membership test operators *in* and *not in* test whether the sequence has a given item or not.

For string type objects, `x in y` is True if and only if `x` is a substring of `y`

## I4 IDENTITY OPERATORS

---

- 'is' and 'is not'

15

---

**Thank You**





---

# CS 200I PYTHON PROGRAMMING

## 2 LECTURE OBJECTIVES

---

- Discuss the control instructions in python



# 3 CONTROL INSTRUCTIONS IN PYTHON

---

- Sequential Control instructions
- Decision Control instructions
- Loop Control instructions (Repetition control instructions)

## 4 SEQUENTIAL CONTROL STATEMENTS

---

- Normal flow of program statements
- $a, b = 5, 10$
- $Sum = a + b$
- $Diff = a - b$
- $Quo = a / b$
- $Mod = a \% b$
- `Print("sum=", Sum)`

## 5 DECISION CONTROL STATEMENTS

---

Execution control will change depending on the condition

Most common example is log in into any portal by verifying the username and password

In python, syntax for decision control statement is as follows

```
if(condition):
```

```
 statements
```

```
else:
```

```
 statements
```

There can be independent if()

```
if(condition):
```

```
 statements of if
```

```
Remaining statements of the code
```

## 6 EXPRESSIONS WITHIN IF

---

- Python interpreter will have to evaluate an expression given in the if()
- The output after evaluation would be Boolean (True or False)
- The conditional expression given within the if () can be
  - Comparison expressions
  - Logical expressions
  - Membership expressions
  - Identity expressions

## 7 COMPARISON EXPRESSIONS

---

- $A > B \rightarrow$  returns True if and only if the actual value of A is greater than B ( never considers the absolute values)
- If A and B are strings, the case of the letters will determine True or False
- Uppercase letters have lower ASCII value than the lowercase.
- 'Python' > 'python' is evaluated as False
- 'Python' > 'PYthon' is evaluated as True

## 8 LOGICAL EXPRESSIONS

---

- To combine two or more comparison expressions
- $A > B$  *and*  $A > C$  evaluates to TRUE if A is greater than both B and C
- $A > B$  *and*  $B > C$  can be simplified as  $A > B > C$  in python
- $A < B$  *and*  $A < C$  can be simplified as  $A < B < C$
- $A > B$  *or*  $A > C$  evaluates to TRUE if either A is greater than B or A is greater than C
- *not*  $X > 2$  evaluates to TRUE if X is less than 2



## 9 ALL() AND ANY()

---

- If there are a number of expressions to be combined with *and* we can use built in function **all()**

Similarly, instead of using *or* expressions, we can apply **any()**

**Both all() and any() can take only one argument :- could be a list or tuple**

Example:

instead of writing

```
if(a>0 and b>0 and c>0 and a+b+c==100):
```

we can use

```
if(all((a>0,b>0,c>0,a+b+c==100))):
```

## 10 MEMBERSHIP AND IDENTITY

---

- In, not in, is, is not can also be used within if clause. They will also return True or False
- S= "COMPUTER"
- T="computer"
- I='C'
- I in S evaluates to True
- 'A' in S evaluates to False
- S is T evaluates to False

## II NESTED IF

---

- If there are multiple conditions to be checked, we can use if- elif-else syntax

- `if(condition1):`

- `do this`

- `elif(condition2):`

- `do this`

- `elif(condition3):`

- `do this`

- `else:`

- `do this`

## 12 EXERCISES

---

- Write a program to check for odd and even
- Write a program to check for leap year
- Write a program to check for maximum among three numbers

## I3 CONDITIONAL EXPRESSION

---

- Like the conditional operator in C, python supports single line decision control statement
- The Syntax is  
expression 1 if condition else expression 2

Example:

```
Temp=int(input("enter temperature in Celsius\n"))
```

```
Status="Hot" if Temp>40 else "Normal"
```

## 14 NESTING OF CONDITIONAL EXPRESSIONS

---

- We can have conditional expressions within a conditional expression
- In the previous Example
- Status="Hot" if temperature>40 else "Cold" if temperature <20 else "Normal"



# 15 REPETITION CONTROL INSTRUCTIONS

---

- while
- while with else
- for
- for with else
- Nested for loops

## 16 WHILE

---

- Similar to that of C
- Slight modification in the syntax

`while (condition):`

`statements`

Condition is similar to that in decision control statements

## 17 EXAMPLE

---

- A program to display all the digits from 0 to 10 in ascending order

```
a=0
```

```
while(a<=10):
```

```
 print(a)
```

```
 a+=1
```

## 18 BREAK AND CONTINUE

---

- 'break' is used to exit from the loop
- 'continue' is used to bypass the remaining statements in the loop and to go back to the beginning of the loop

```
while(a>10):
```

```
 print(a)
```

```
 if(a==2):
```

```
 break
```

## 19 PASS STATEMENT

---

Unlike C, python does not support empty control statements

```
if(condition)
```

```
{
```

```
}
```

is valid in C whereas

`if(condition):` without any statements is invalid in Python

Similarly `while(condition):` give you syntax error

Therefore Python supports 'pass' statements within a control statement

## 20 PASS STATEMENT

---

- It means do nothing if the condition is true
- It can be a placeholder for future code
- 'pass' can be included in if(), while(), for() and in else statement
- Example:
- if(n>0):

    print(n\*n)

elif(n<0):

    print(n\*n\*n)

else:

    pass



## 21 WHILE WITH ELSE

---

The additional feature of while statement in Python is the inclusion of else block

```
while(condition):
```

```
 statements
```

```
else:
```

```
 statements
```

Else block will be executed when the condition given in while turns to be False. ( Upon completion of the iterations of while loop)

## 22 EXAMPLE – PRIME CHECKING

---

```
n=int(input("Enter any number"))
```

```
h=n//2
```

```
while h>=2:
```

```
 if n%h==0:
```

```
 print("composite\n")
```

```
 break
```

```
 h=h-1
```

```
else:
```

```
 print("number is prime\n")
```



## 23 FOR LOOP

---

- Most commonly used statement for iterations

Syntax is different from C

Increment and decrement operators are not available.

In Python, iteration is done either with the help of any iterable object or with built in functions

String, list, tuple, dictionary etc are commonly used within for loop for iteration

## 24 EXAMPLE

---

- S="BOLD"

for j in S:

    print(j)

OUTPUT

B

O

L

D

Note: j takes the values from B to D and the loop iterates till the end of the string

## 25 ENUMERATE

---

- To see the index of each element, built in function enumerate can be used in for loop
- for j in enumerate(S):  
    print(i)

OUTPUT

(0,B)

(1,O)

(2,L)

(3,D)

## 26 RANGE()

---

In the absence of iterable objects, range() function can be used for definite number of iterations.

Syntax is

```
range(start, stop, step)
```

Range can take integers ( positive and negative) but not floating point numbers.

By default start is zero, by default step is 1

Range should always have a stop



## 27 EXAMPLE

---

- Program to display numbers from 0 to 10

```
for i in range(0,5):
 print(i)
```

OUTPUT

0

1

2

3

4

## 28 RANGE() EXAMPLES

---

- for j in range(-5,5,2):  
    print(j)
- for j in range(10):  
    print(j)
- for j in range(10,5,-2):  
    print(j)

## 29 FOR WITH ELSE

---

- Like while statement for loop can also include an else block
- Else block will be executed upon the condition in for loop becomes invalid.
- Usually for with else may have if() and break within for loop
- for j in S:

    if(condition):

        do this

        break

else:



## 30 EXAMPLE -VOWELSCHECK

---

- Write a python script to check whether all characters present in the string are vowels
- `S=input("enter any string\n")`
- `vowels="AEIOUaeiou"`  
    for j in S:  
        if(j not in vowels):  
            print("False")  
            break  
    else:  
        print("True")

## 3 | NESTED FOR LOOP

---

- Python supports nesting of for loops
- S="JUPYTER"
- T="12345"
- for j in S:  
    for k in T:  
        print(j,k)

Nested for loops are normally used for analysing single or multidimensional arrays

**Thank You**

