Computing Fundamentals using Python

SUBJECT CODE : UQ25CA151A

Samyukta D Kumta

Computer Applications

Introduction to Methods

- A method is a block of code that performs a specific task and can be reused multiple times.
- In Python, methods are functions defined inside classes, but built-in data types (like lists, tuples, sets, and dictionaries) also come with predefined methods.

Methods help in:

- Reducing code repetition.
- Improving readability.
- Allowing modular programming.

Methods in Lists

A **list** is an ordered, mutable collection of elements.

Method	Description	Example
append(x)	Adds an element at the end	lst.append(10)
insert(i, x)	Inserts at index i	lst.insert(1, 20)
remove(x)	Removes first occurrence of x	lst.remove(20)
pop(i)	Removes and returns element at index i (default last)	lst.pop()
sort()	Sorts list in ascending order	lst.sort()
reverse()	Reverses order of list	lst.reverse()

Methods in Lists

```
# Creating a list
fruits = ["apple", "banana", "cherry"]
print("Original List:", fruits)

# append() - Add element at the end
fruits.append("mango")
print("After append:", fruits)

# insert() - Insert element at specific index
fruits.insert(1, "orange")
print("After insert:", fruits)
```

Methods in Lists

```
# remove() - Remove first occurrence of element
fruits.remove("banana")
print("After remove:", fruits)

# pop() - Remove element by index (default last element)
popped_item = fruits.pop()
print("After pop:", fruits)
print("Popped item:", popped_item)

# extend() - Add elements from another list
fruits.extend(["grape", "pineapple"])
print("After extend:", fruits)
```

Methods in Lists

```
# sort() - Sort list in ascending order
fruits.sort()
print("After sort:", fruits)

# reverse() - Reverse the list
fruits.reverse()
print("After reverse:", fruits)

# index() - Find index of an element
print("Index of 'grape':", fruits.index("grape"))
```

Methods in Lists

```
# count() - Count occurrences of an element
fruits.append("apple")
print("After adding another apple:", fruits)
print("Count of 'apple':", fruits.count("apple"))

# clear() - Remove all elements
fruits.clear()
print("After clear:", fruits)
```

Methods in Tuples

A tuple is an ordered, immutable collection of elements. It has limited methods since it cannot be changed.

Method	Description	Example
count(x)	Counts occurrences of x	t.count(2)
index(x)	Returns index of first occurrence	t.index(3)

Operations on Tuples

Operation	Description	Example	Output
Slicing	Access part of tuple using indexes	t[1:4]	(20, 30, 20)
Concatenation	Join two tuples	(1,2) + (3,4)	(1,2,3,4)
Repetition	Repeat tuple elements	("Hi",) * 3	('Hi', 'Hi', 'Hi')
Membership	Check if an element exists	20 in t	True
len()	Find length of tuple	len(t)	4
max()	Largest element (if numeric)	max((5,10,2))	10
min()	Smallest element (if numeric)	min((5,10,2))	2
sum()	Sum of elements (if numeric)	sum((5,10,2))	17

Methods in Tuples

```
# Creating a tuple
numbers = (10, 20, 30, 20, 40, 50)
print("Original Tuple:", numbers)

# count() - Counts the number of times a value appears
count_20 = numbers.count(20)
print("Count of 20:", count_20)

# index() - Returns the index of first occurrence of a value
index_30 = numbers.index(30)
print("Index of 30:", index_30)
```

Methods in Tuples

```
# Using tuple concatenation
new_tuple = numbers + (60, 70)
print("After concatenation:", new_tuple)

# Using tuple repetition
repeat_tuple = ("Hi",) * 3
print("After repetition:", repeat_tuple)
```

Methods in Sets

A set is an unordered collection of unique elements.

Method	Description	Example
add(x)	Adds element	s.add(5)
remove(x)	Removes element, error if not found	s.remove(2)
discard(x)	Removes element if present	s.discard(10)
union(t)	Returns union	s.union(t)
intersection(t)	Returns intersection	s.intersection(t)
difference(t)	Returns difference	s.difference(t)

Methods in Sets

```
# Creating a set
numbers = {1, 2, 3, 4}
print("Original Set:", numbers)

# add() - Add an element
numbers.add(5)
print("After add:", numbers)

# remove() - Remove element (Error if not found)
numbers.remove(3)
print("After remove(3):", numbers)
```

Methods in Sets

```
# union() - Combine sets
set_a = {1, 2, 3}
set_b = {3, 4, 5}
print("Union:", set_a.union(set_b))

# intersection() - Common elements
print("Intersection:", set_a.intersection(set_b))
```

Methods in Sets

```
# discard() - Remove element (No error if not found)
numbers.discard(10)
print("After discard(10):", numbers)

# pop() - Remove and return a random element
popped_item = numbers.pop()
print("After pop:", numbers)
print("Popped item:", popped_item)
```

Methods in Sets

```
# difference() - Elements in set_a but not in set_b
print("Difference (A-B):", set_a.difference(set_b))

# symmetric_difference() - Elements in either but not both
print("Symmetric Difference:", set_a.symmetric_difference(set_b))

# update() - Add elements of another set
numbers.update({7, 8})
print("After update:", numbers)

# clear() - Remove all elements
numbers.clear()
print("After clear:", numbers)
```

Methods in Sets

```
# difference() - Elements in set_a but not in set_b
print("Difference (A-B):", set_a.difference(set_b))

# symmetric_difference() - Elements in either but not both
print("Symmetric Difference:", set_a.symmetric_difference(set_b))

# update() - Add elements of another set
numbers.update({7, 8})
print("After update:", numbers)

# clear() - Remove all elements
numbers.clear()
print("After clear:", numbers)
```

Methods in Dictionary

A **dictionary** is an unordered collection of key-value pairs.Keys are unique and immutable, values can be anything.

Method	Description	Example
keys()	Returns all keys	d.keys()
values()	Returns all values	d.values()
items()	Returns key-value pairs	d.items()
get(key)	Returns value if key exists, else None	d.get("age")
update(d2)	Updates dictionary with another	d.update({"city":"Delhi"})
pop(key)	Removes key-value pair	d.pop("age")

Methods in Dictionary

```
# Creating a dictionary
student = {"name": "Sam", "age": 21, "grade": "A"}
print("Original Dictionary:", student)

# get() - Returns the value for a key
print("Get age:", student.get("age"))
print("Get city (not present):", student.get("city"))  # returns None
```

Methods in Dictionary

```
# keys() - Returns all keys
print("Keys:", student.keys())

# values() - Returns all values
print("Values:", student.values())

# items() - Returns all key-value pairs
print("Items:", student.items())

# update() - Add or update key-value pairs
student.update({"city": "Bangalore", "grade": "A+"})
print("After update:", student)
```

Methods in Dictionary

```
# pop() - Removes and returns a value by key
removed_age = student.pop("age")
print("After pop(age):", student)
print("Removed age:", removed_age)

# popitem() - Removes and returns last inserted key-value pair
removed_item = student.popitem()
print("After popitem():", student)
print("Removed item:", removed_item)
```

Methods in Dictionary

```
# setdefault() - Returns value if key exists, else adds key with default value
student.setdefault("course", "Computer Science")
print("After setdefault:", student)

# copy() - Shallow copy of dictionary
student_copy = student.copy()
print("Copied Dictionary:", student_copy)

# clear() - Remove all items
student.clear()
print("After clear:", student)
```

Methods in Dictionary

```
# setdefault() - Returns value if key exists, else adds key with default value
student.setdefault("course", "Computer Science")
print("After setdefault:", student)

# copy() - Shallow copy of dictionary
student_copy = student.copy()
print("Copied Dictionary:", student_copy)

# clear() - Remove all items
student.clear()
print("After clear:", student)
```

THANK YOU

Samyukta D Kumta
Department of Computer Applications
samyuktad@pes.edu