Initializing to max or min:

maxSum = float('-inf')

Input:

name=input("enter your name: ")  
weight = float(input("Enter your weight in kg: "))

Output:

print(f"I work in {company} and my age is {age}")  
print("Your BMI is:", bmi)

Casting:

print("Index of 'a':" + str(index))

for val in the list str1 – if its alphabet or num – it will be kept or else removed – and create a list

list([val for val in str1 if val.isalnum()])

Reverse:

str2 == str2[::-1]

Join:

x = "separator".join(myTuple)

Range:

for i in range(0, len(nums)-1):

for i in range(len(new\_str)):

for num in list\_num:

for char in strin:

Length:

len(object)

Enumarate:

for i, num in enumerate(nums):

Other:

Max()

Min()

Pow():

def pow(base: int, exp: \_PositiveInteger, mod: None = None) -> int

Round()

Sorted:

def sorted(\_\_iterable: Iterable[\_T], \*, key: Callable[[\_T], SupportsRichComparison], reverse: bool = False) -> list[\_T]

dic\_sort = dict(sorted(dic.items(), key = lambda x: x[1], reverse = True))

String:

def capitalize(self: LiteralString) -> LiteralString

def count(self, x: str, \_\_start: SupportsIndex | None = ..., \_\_end: SupportsIndex | None = ...) -> int

def endswith(  
 self, \_\_suffix: str | tuple[str, ...], \_\_start: SupportsIndex | None = ..., \_\_end: SupportsIndex | None = ...  
) -> bool

def find(self, \_\_sub: str, \_\_start: SupportsIndex | None = ..., \_\_end: SupportsIndex | None = ...) -> int

def index(self, \_\_sub: str, \_\_start: SupportsIndex | None = ..., \_\_end: SupportsIndex | None = ...) -> int

def isalnum(self) -> bool  
def isalpha(self) -> bool   
def isascii(self) -> bool  
def isdecimal(self) -> bool  
def isdigit(self) -> bool  
def isidentifier(self) -> bool  
def islower(self) -> bool  
def isnumeric(self) -> bool  
def isprintable(self) -> bool  
def isspace(self) -> bool  
def istitle(self) -> bool  
def isupper(self) -> bool

def lower(self: LiteralString) -> LiteralString

l before function is left side strip:

lstrip(self: LiteralString, \_\_chars: LiteralString | None = None) -> LiteralString

def strip(self: LiteralString, \_\_chars: LiteralString | None = None) -> LiteralString

def replace(  
 self: LiteralString, \_\_old: LiteralString, \_\_new: LiteralString, \_\_count: SupportsIndex = -1  
) -> LiteralString

r before the function is to return only last accurence or index:

def rfind(self, \_\_sub: str, \_\_start: SupportsIndex | None = ..., \_\_end: SupportsIndex | None = ...)

def split(self: LiteralString, sep: LiteralString | None = None, maxsplit: SupportsIndex = -1) -> list[LiteralString]

def splitlines(self: LiteralString, keepends: bool = False) -> list[LiteralString]

def startswith(  
 self, \_\_prefix: str | tuple[str, ...], \_\_start: SupportsIndex | None = ..., \_\_end: SupportsIndex | None = ...  
) -> bool

def upper(self: LiteralString) -> LiteralString:

List:

def copy(self) -> list[\_T]:

def append(self, \_\_object: \_T) -> None

def extend(self, \_\_iterable: Iterable[\_T]) -> None

def pop(self, \_\_index: SupportsIndex = -1) -> \_T

removes last or if you want to remove specific then:

team\_elements.pop(ipl.index(team\_name))

def remove(self, \_\_value: \_T) -> None

remove first occurrence of the element specified

def index(self, \_\_value: \_T, \_\_start: SupportsIndex = 0, \_\_stop: SupportsIndex = sys.maxsize) -> int

def count(self, \_\_value: \_T) -> int

def insert(self, \_\_index: SupportsIndex, \_\_object: \_T) -> None

def sort(self: list[SupportsRichComparisonT], \*, key: None = None, reverse: bool = False) -> None

Dictonary:

new\_dic = {}  
new\_dic['name'] = "sourav"

def copy(self) -> dict[\_KT, \_VT]:

def keys(self) -> dict\_keys[\_KT, \_VT]

def values(self) -> dict\_values[\_KT, \_VT]

def items(self) -> dict\_items[\_KT, \_VT]

def fromkeys(cls, \_\_iterable: Iterable[\_T], \_\_value: None = None) -> dict[\_T, Any | None]

def get(self, \_\_key: \_KT) -> \_VT | None

dic[char] = dic.get(char,0)+1 (if not present put 0 for value for the given key)

def pop(self, \_\_key: \_KT) -> \_VT

Set:

def add(self, \_\_element: \_T) -> None

def copy(self) -> set[\_T]

def difference(self, \*s: Iterable[Any]) -> set[\_T]

def difference\_update(self, \*s: Iterable[Any]) -> None

def discard(self, \_\_element: \_T) -> None

def intersection(self, \*s: Iterable[Any]) -> set[\_T]

def intersection\_update(self, \*s: Iterable[Any]) -> None

def isdisjoint(self, \_\_s: Iterable[Any]) -> bool

def issubset(self, \_\_s: Iterable[Any]) -> bool

def issuperset(self, \_\_s: Iterable[Any]) -> bool

def remove(self, \_\_element: \_T) -> None

def union(self, \*s: Iterable[\_S]) -> set[\_T | \_S]

def update(self, \*s: Iterable[\_T]) -> None

Slice:

class slice:  
 @property  
 def start(self) -> Any  
 @property  
 def step(self) -> Any  
 @property  
 def stop(self) -> Any

Tuple:

count(self, \_\_value: Any) -> int

index(self, \_\_value: Any, \_\_start: SupportsIndex = 0, \_\_stop: SupportsIndex = sys.maxsize) -> int

Counter: it is a sub class where we can get a dic of key and its accurence in list or stirring ect

To return max accruing at list

arr=Counter(arr)

arr=sorted(arr.items(), key=lambda x: x[1] ,reverse=True)

return arr[0][0]