Lab Work — Lambda Function

Objective

This lab teaches how to use Python lambda functions for concise operations. You will practice creating basic lambdas, using multiple arguments, combining lambdas with map(), filter(), sorted(), and reduce(), and replacing small def functions. The lab demonstrates how lambdas simplify code for quick transformations and calculations.

Task 1 — Basic Lambda

Create simple lambda for one operation.

Test with 2 inputs.

Compare with normal function.

In [9]:
```
add = lambda a: a+5
print(add(12))
print(add(10))

#normal function do the same thing but comprises of multiple lines where as la
def add_func(x):
    return x+5
print(add_func(12))
print(add_func(10))
```

```
17
15
17
15
```

Task 2 — Lambda With Multiple Arguments

Lambda with 2 inputs.

Lambda with 3 inputs.

Try keyword-only argument in lambda.

In [12]:
```
two_inputs= lambda a,b: a+b
print(two_inputs(3,7))
print(two_inputs(5,25))

three_inputs=lambda x,y,z:x*y*z
print(three_inputs(2,2,2))
print(three_inputs(4,5,6))
print(three_inputs(3,3,1))

keyword_lambda= lambda x, *, y: x+y
```

```
print(keyword_lambda(5,y=10))
print(keyword_lambda(98,y=2))
```

```
10
30
8
120
9
15
100
```

Task 3 — Lambda With map()

Transform list of numbers.

Transform list of strings.

Transform list of tuples.

In [15]:
```
numbers=[1,2,3,4,5]
print(list(map(lambda x: x+2,numbers)))

words=["foo","bar","baz"]
print(list(map(lambda b: b.upper(),words)))

ListOfTuples= [(1,2),(3,4),(5,6)]
print(list(map(lambda d: d[0]*d[1],ListOfTuples)))
```

```
[3, 4, 5, 6, 7]
['FOO', 'BAR', 'BAZ']
[2, 12, 30]
```

Task 4 — Lambda With filter()

Keep only even numbers.

Keep strings longer than 3 characters.

Filter mixed-type list.

In [22]:
```
num =[1,2,3,4,5,6,7,8,9,10]
even = list(filter(lambda x: x%2==0,num))
print(even)

words=["Azkaa","book","table","sick","dog","sun"]
word=list(filter(lambda w: len(w)>3,words))
print(word)

mixed = [1, "apple", 3.5, "hi", 10, True]
only_nums= list(filter(lambda x:type(x)in (int,str), mixed))
print(only_nums)
```

```
[2, 4, 6, 8, 10]
['Azkaa', 'book', 'table', 'sick']
[1, 'apple', 'hi', 10]
```

Task 5 — Lambda With sorted() key

Sort list by length.

Sort list of tuples by second element.

Sort dictionary items by values.

In [1]:
```python
words = ["apple", "kiwi", "banana", "fig"]
print(sorted(words, key=lambda w: len(w)))

b=[(1, 5), (3, 1), (2, 9)]
print(sorted(b,key=lambda z: z[1]))

a={"Ali": 89,"sara":500, "hina":100}
print(sorted(a.items(), key= lambda x: x[1]))
```

```
['fig', 'kiwi', 'apple', 'banana']
[(3, 1), (1, 5), (2, 9)]
[('Ali', 89), ('hina', 100), ('sara', 500)]
```

Task 6 — Lambda With reduce()

Combine numbers in a list.

Join multiple strings.

Find custom smallest or largest element.

In [32]:
```python
from functools import reduce

numbers = [1, 2, 3, 4, 5]
print(reduce(lambda x, y: x + y, numbers))

words = ["I", "love", "Python"]
print(reduce(lambda a, b: a + " " + b, words))

numbers2 = [7, 2, 9, 4]
print(reduce(lambda a, b: a if a < b else b, numbers2))

print(reduce(lambda a, b: a if a > b else b, numbers2))
```

```
15
I love Python
2
9
```

Task 7 — Replace small def with lambda

Rewrite function: def label_meter(id): return "Meter-" + str(id).zfill(3)

Call both with 12.

Compare readability in one sentence.

In [33]:
```
def label_meter(id):
    return "Meter-" + str(id).zfill(3)


label_meter = lambda id: "Meter-" + str(id).zfill(3)

print(label_meter(12))
print(label_meter(12))

#The def version is clearer because it is more readable and easier to understa
```

```
Meter-012
Meter-012
```

Task 8 — Sort electrical components using lambda key

Sort components = [("LED",5),("Fan",60),("Motor",350),("Bulb",40)] by watt rating using sorted() + lambda.

Repeat using operator.itemgetter.

State one advantage of itemgetter.

In [ ]:
```
components = [("LED", 5), ("Fan", 60), ("Motor", 350), ("Bulb", 40)]

sorted_lambda = sorted(components, key=lambda x: x[1])
print(sorted_lambda)


from operator import itemgetter
sorted_itemgetter = sorted(components, key=itemgetter(1))
print(sorted_itemgetter)

#Advantage of itemgetter:
#Faster and more readable than lambda for simple element access
```

```
[('LED', 5), ('Bulb', 40), ('Fan', 60), ('Motor', 350)]
[('LED', 5), ('Bulb', 40), ('Fan', 60), ('Motor', 350)]
```

Task 9 — Convert sensor strings using map() + lambda

Convert readings = ["2.5","5.0","3.3"] to floats + 0.1 offset using map() + lambda.

Show equivalent list comprehension.

Say which is cleaner.

In [40]:
```python
readings = ["2.5", "5.0", "3.3"]
calibrated = list(map(lambda x: float(x) + 0.1, readings))
print(calibrated)

calibrated_lc = [float(x) + 0.1 for x in readings]
print(calibrated_lc)

#List comprehension is shorter and easier to read for small operations.
```

```
[2.6, 5.1, 3.4]
[2.6, 5.1, 3.4]
```

Task 10 — Filter low power values using filter() + lambda

Keep only powers = [20,75,45,120,10] above 50 W using filter().

Show equivalent list comprehension.

In [ ]:
```python
powers = [20, 75, 45, 120, 10]

high_power = list(filter(lambda x: x > 50, powers))
print(high_power)

high_power_lc = [x for x in powers if x > 50]
print(high_power_lc)
```

```
[75, 120]
[75, 120]
```

Task 11 — Use reduce() to combine values

Compute product of Z = [2,3,4] using reduce() + lambda.

Show simpler alternative with loop or math.prod.

Write one line on which is easier.

In [43]:
```python
from functools import reduce
import math

Z = [2, 3, 4]
total_impedance = reduce(lambda x, y: x * y, Z)
print(total_impedance)

product = 1
for z in Z:
    product *= z
print(product)

print(math.prod(Z))
```

```
24
24
24
```

Task 12 — Conditional lambda for load classification

Lambda returns "High Load" if power > 100 W else "Low/Normal Load".

Test with 150 W and 90 W.

In [42]:
```
load_class = lambda power: "High Load" if power > 100 else "Low/Normal Load"

print(load_class(150))
print(load_class(90))
```

```
High Load
Low/Normal Load
```