Python-lab

May 3, 2026

0.1 Input Reading

[1]:
```
input()
```

```
 Hello, world!
```

[1]:
```
'Hello, world!'
```

[2]:
```
input().split()
```

```
 Hello, world!
```

[2]:
```
['Hello,', 'world!']
```

[6]:
```
name = input()
name_list = name.split()
first = name_list[0]
last = name_list[1]
print("First name:", first, "and last name:", last)
```

```
 Zoe Paraskevopoulou
```

```
First name: Zoe and last name: Paraskevopoulou
```

[7]:
```
first, last = input().split()
print("First name:", first, "and last name:", last)
```

```
 Zoe Paraskevopoulou
```

```
First name: Zoe and last name: Paraskevopoulou
```

[4]:
```
n1, n2 = input().split()
print(n1 + n2)
```

```
 17 42
```

```
1742
```

[5]:
```
n1, n2 = input().split()
print(int(n1) + int(n2))
```

```
 17 42
```

```
59
```

[ ]:
```
first, second = input().split()
print(int(first) + int(second))
```

This works, but it is not idiomatic Python. Also, it doesn't generalize for more than two numbers.

How can we convert a list of strings arbitrary length to a list of integers?

[8]:
```
# this works but it is not very "Pythonic"
l = []

inp = input().split()

for x in inp: l.append(int(x))

print(l)
```

```
 1 2 3 4
```

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

[9]:
```
[int(x) for x in input().split()]
```

```
 1 2 3 4
```

[9]:
```
[1, 2, 3, 4]
```

0.2 Mapping Over Collections

[10]:
```
L = [1, 2, 3, 4]

def f(x):
    return x*10

list(map(f, L))
```

[10]:
```
[10, 20, 30, 40]
```

[11]:
```
list(map(lambda x: x * 10, L))
```

[11]:
```
[10, 20, 30, 40]
```

[12]:
```
type(map(f, L))
```

[12]:
```
map
```

Map objects are similar to generators.

[14]:
```
m = map(f, L)
next(m)
```

[14]:
```
10
```

[15]:
```
next(m)
```

[15]: 20

Quiz: How can we write the same thing with a generator?

Both of the following programs read a list of integers from the standard input and print them to the standard output.

[16]:
```
print(sum(map(int, input().split())))
```

```
 1 2 3 4 5
```

```
15
```

[17]:
```
print(sum(int(x) for x in input().split()))
```

```
 1 2 3 4 5
```

```
15
```

0.3 Exercise 1

Given two sequences of numbers, print all the numbers that belong to the first sequence but not the second one, in the order they appear in the first list.

Example Input

4 9 5 1 10

5 7 2 4 1

Example Output

9 10

[19]:
```
S1 = list(map(int, input().split()))
S2 = list(map(int, input().split()))
```

```
 4 9 5 1 10
 5 7 2 4 1
```

[20]:
```
S1
```

[20]: [4, 9, 5, 1, 10]

[21]:
```
S2
```

[21]: [5, 7, 2, 4, 1]

[22]:
```
for x in S1:
    if x not in S2:
        print(x)
```

```
9
10
```

[25]:
```
L = []

for x in S1:
    if not x in S2:
        L.append(x)
print(L)
```

```
[9, 10]
```

[27]:
```
# similar, but more Pythonic
L = [x for x in S1 if not x in S2]

print(L)
```

```
[9, 10]
```

[28]:
```
" ".join(["hi", "there"])
```

[28]:
```
'hi there'
```

[29]:
```
print(" ".join(str(x) for x in S1 if not x in S2))
```

```
9 10
```

[30]:
```
print(*(str(x) for x in S1 if not x in S2))
```

```
9 10
```

What is the running time of the above code?

[31]:
```
S1 = list(map(int, input().split()))
S2 = set(map(int, input().split()))
```

```
 4 9 5 1 10
5 7 2 4 1
```

[32]:
```
print(*(str(x) for x in S1 if x not in S2))
```

```
9 10
```

0.4 Exercise 2

Е П Δ П :

М , , . Т
, . Г
.

N , , .

Example Input: 5 6 4 6 3 4 1

Example Output: 3

[ ]:
```
a = list(map(int, input().split()))
```

[ ]:
```
print(a)
```

[ ]:
```
def solve(a):
    N = len(a)
    count = 0
    for i in range(N):
        tallest = True
        for j in range (i+1, N):
            if a[i] <= a[j]:
                tallest = False
                break
        if tallest: count += 1
    return count
```

[ ]:
```
solve(a)
```

[ ]:
```
def solve_fast(a):
    N = len(a)
    tallest = 0
    count = 0

    for x in (a):
        if tallest < x:
            tallest = x
            count += 1

    return count
```

[ ]:
```
def solve_fast(a):
    N = len(a)
    tallest = 0
    count = 0

    for x in reversed(a):
        if tallest < x:
            tallest = x
            count += 1

    return count
```

[ ]:
```
solve_fast(a)
```

Let's test that the two solutions compute the same result:

```
[ ]: import random
```

```
[ ]: random
```

```
[ ]: type(random)
```

```
[ ]: random.randrange(100)
```

```
[ ]: random.randrange(100, 199)
```

```
[ ]: N = random.randrange(1,1000)
```

```
[ ]: a = [random.randrange(1,100) for i in range(N)]
```

```
[ ]: print(N)
```

```
[ ]: print(a)
```

```
[ ]: solve(a)
```

```
[ ]: solve_fast(a)
```

```
[ ]: for _ in range(1000):
         N = random.randrange(1,100)
         a = [random.randrange(1,100) for i in range(N)]
         if solve_fast(a) != solve(a): print("BOOM!", a)
     print("Done!")
```

0.5 Itertools

```
[ ]: import itertools
```

```
[ ]: list(itertools.permutations([1,2,3]))
```

```
[ ]: # Iterate through all permutations of a collection

     for x in itertools.permutations([1,2,3]):
         print(x)
```

```
[ ]: daltons = ["Joe", "Jack", "William", "Averel"]
```

```
[ ]: for x in itertools.combinations(daltons, 3):
         print(x)
```

```
[ ]: list(itertools.combinations([1,2,3,4], 4))
```

These are helpful for bruteforce solutions, but be careful as they can be very slow.

0.6 Exercise 3

Given a set of elements, partition it into two subsets such that the sum of the two subsets is the same.

Example Input

1 2 3 4 2

Example Output

2 4

[ ]:
```
L = list(map(int, input().split()))
```

[ ]:
```
L
```

[ ]:
```
def partition(l):
    S = sum(l)

    for n in range(1, len(l)//2 + 1):
        for candidate_sol in itertools.combinations(l, n):
            if 2*sum(candidate_sol) == S:
                return candidate_sol
```

[ ]:
```
partition(L)
```

[ ]:
```
L = [1, 4, 3, 8, 19, 12, 27]
partition(L)
```

[ ]:
```
partition(L) == None # No solution for this input
```

[ ]:
```
partition([1, 2, 3, 4, 2])
```