User Defined Functions

January 25, 2023

1 User Defined Function

- Predefined Functions (Built_in functions)
- User Defined Functions

1.1 Syntax to create a user defined function

1.1.1 Function definition

```
def function_name(parameters):
    block of statements
```

1.1.2 Function Call

```
function_name(arguments)
```

1.1.3 Types of functions

- Functions without arguments without return type
- Functions with arguements without return type
- Functions with arguments with return type
- Functions without arguments with return type

Functions without arguments without return type

[18]:
```
# Fun def
def wish(): # withour arguments
    # if you don't see return statement anywhere in the function body
    # it is a function without return type
    print("Hello all, welcome back to this session")
# Fun call
wish()
wish()
wish()
wish()
```

```
Hello all, welcome back to this session
Hello all, welcome back to this session
Hello all, welcome back to this session
Hello all, welcome back to this session
```

Functions with arguments without return type

[14]:
```python
# Function Definition
def add(a, b): # with arguments but withour return type
    print(a + b)

# Function call
add(10, 20)
add(100, 200)
add(1000, 2000)
```

```
30
300
3000
```

[15]:
```python
def set_grade(stu_per):
    if stu_per >= 90:
        print('O')
    elif stu_per >= 80:
        print('A')
    elif stu_per >= 70:
        print('B')
    elif stu_per >= 60:
        print('C')
    elif stu_per >= 50:
        print('D')
    else:
        print('E')

s1 = 78.4
s2 = 46.7
s3 = 88.9
set_grade(s1)
set_grade(s2)
set_grade(s3)
```

```
B
E
A
```

[19]:
```python
# Function Definition
def add(a, b): # with arguments but withour return type
    print(a + b)

# Function call
add(10, 20)
```

```
30
```

[20]:
```
# Function Definition
def add(a, b): # with arguments but withour return type
    print(a + b)

# Function call
add(10) # invalid function call to add()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [20], in <cell line: 6>()
      3     print(a + b)
      5 # Function call
----> 6 add(10)

TypeError: add() missing 1 required positional argument: 'b'
```

[21]:
```
# Function Definition
def add(a, b): # with arguments but withour return type
    print(a + b)

# Function call
add(10, 20, 30) # invalid function call to add()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [21], in <cell line: 6>()
      3     print(a + b)
      5 # Function call
----> 6 add(10, 20, 30)

TypeError: add() takes 2 positional arguments but 3 were given
```

[26]:
```
def print_chey_bey(start, stop):
    for i in range(start, stop + 1):
        print(i, end = ' ')

a, b = 10, 20
print_chey_bey(a, b)
x, y = 100, 200
print()
print_chey_bey(x, y)
p, q = 1000, 2000
print()
print_chey_bey(p, q)
```

```
10 11 12 13 14 15 16 17 18 19 20
```

100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139
140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159
160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199
200
1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014 1015
1016 1017 1018 1019 1020 1021 1022 1023 1024 1025 1026 1027 1028 1029 1030 1031
1032 1033 1034 1035 1036 1037 1038 1039 1040 1041 1042 1043 1044 1045 1046 1047
1048 1049 1050 1051 1052 1053 1054 1055 1056 1057 1058 1059 1060 1061 1062 1063
1064 1065 1066 1067 1068 1069 1070 1071 1072 1073 1074 1075 1076 1077 1078 1079
1080 1081 1082 1083 1084 1085 1086 1087 1088 1089 1090 1091 1092 1093 1094 1095
1096 1097 1098 1099 1100 1101 1102 1103 1104 1105 1106 1107 1108 1109 1110 1111
1112 1113 1114 1115 1116 1117 1118 1119 1120 1121 1122 1123 1124 1125 1126 1127
1128 1129 1130 1131 1132 1133 1134 1135 1136 1137 1138 1139 1140 1141 1142 1143
1144 1145 1146 1147 1148 1149 1150 1151 1152 1153 1154 1155 1156 1157 1158 1159
1160 1161 1162 1163 1164 1165 1166 1167 1168 1169 1170 1171 1172 1173 1174 1175
1176 1177 1178 1179 1180 1181 1182 1183 1184 1185 1186 1187 1188 1189 1190 1191
1192 1193 1194 1195 1196 1197 1198 1199 1200 1201 1202 1203 1204 1205 1206 1207
1208 1209 1210 1211 1212 1213 1214 1215 1216 1217 1218 1219 1220 1221 1222 1223
1224 1225 1226 1227 1228 1229 1230 1231 1232 1233 1234 1235 1236 1237 1238 1239
1240 1241 1242 1243 1244 1245 1246 1247 1248 1249 1250 1251 1252 1253 1254 1255
1256 1257 1258 1259 1260 1261 1262 1263 1264 1265 1266 1267 1268 1269 1270 1271
1272 1273 1274 1275 1276 1277 1278 1279 1280 1281 1282 1283 1284 1285 1286 1287
1288 1289 1290 1291 1292 1293 1294 1295 1296 1297 1298 1299 1300 1301 1302 1303
1304 1305 1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318 1319
1320 1321 1322 1323 1324 1325 1326 1327 1328 1329 1330 1331 1332 1333 1334 1335
1336 1337 1338 1339 1340 1341 1342 1343 1344 1345 1346 1347 1348 1349 1350 1351
1352 1353 1354 1355 1356 1357 1358 1359 1360 1361 1362 1363 1364 1365 1366 1367
1368 1369 1370 1371 1372 1373 1374 1375 1376 1377 1378 1379 1380 1381 1382 1383
1384 1385 1386 1387 1388 1389 1390 1391 1392 1393 1394 1395 1396 1397 1398 1399
1400 1401 1402 1403 1404 1405 1406 1407 1408 1409 1410 1411 1412 1413 1414 1415
1416 1417 1418 1419 1420 1421 1422 1423 1424 1425 1426 1427 1428 1429 1430 1431
1432 1433 1434 1435 1436 1437 1438 1439 1440 1441 1442 1443 1444 1445 1446 1447
1448 1449 1450 1451 1452 1453 1454 1455 1456 1457 1458 1459 1460 1461 1462 1463
1464 1465 1466 1467 1468 1469 1470 1471 1472 1473 1474 1475 1476 1477 1478 1479
1480 1481 1482 1483 1484 1485 1486 1487 1488 1489 1490 1491 1492 1493 1494 1495
1496 1497 1498 1499 1500 1501 1502 1503 1504 1505 1506 1507 1508 1509 1510 1511
1512 1513 1514 1515 1516 1517 1518 1519 1520 1521 1522 1523 1524 1525 1526 1527
1528 1529 1530 1531 1532 1533 1534 1535 1536 1537 1538 1539 1540 1541 1542 1543
1544 1545 1546 1547 1548 1549 1550 1551 1552 1553 1554 1555 1556 1557 1558 1559
1560 1561 1562 1563 1564 1565 1566 1567 1568 1569 1570 1571 1572 1573 1574 1575
1576 1577 1578 1579 1580 1581 1582 1583 1584 1585 1586 1587 1588 1589 1590 1591
1592 1593 1594 1595 1596 1597 1598 1599 1600 1601 1602 1603 1604 1605 1606 1607
1608 1609 1610 1611 1612 1613 1614 1615 1616 1617 1618 1619 1620 1621 1622 1623
1624 1625 1626 1627 1628 1629 1630 1631 1632 1633 1634 1635 1636 1637 1638 1639
1640 1641 1642 1643 1644 1645 1646 1647 1648 1649 1650 1651 1652 1653 1654 1655
1656 1657 1658 1659 1660 1661 1662 1663 1664 1665 1666 1667 1668 1669 1670 1671

```
1672 1673 1674 1675 1676 1677 1678 1679 1680 1681 1682 1683 1684 1685 1686 1687
1688 1689 1690 1691 1692 1693 1694 1695 1696 1697 1698 1699 1700 1701 1702 1703
1704 1705 1706 1707 1708 1709 1710 1711 1712 1713 1714 1715 1716 1717 1718 1719
1720 1721 1722 1723 1724 1725 1726 1727 1728 1729 1730 1731 1732 1733 1734 1735
1736 1737 1738 1739 1740 1741 1742 1743 1744 1745 1746 1747 1748 1749 1750 1751
1752 1753 1754 1755 1756 1757 1758 1759 1760 1761 1762 1763 1764 1765 1766 1767
1768 1769 1770 1771 1772 1773 1774 1775 1776 1777 1778 1779 1780 1781 1782 1783
1784 1785 1786 1787 1788 1789 1790 1791 1792 1793 1794 1795 1796 1797 1798 1799
1800 1801 1802 1803 1804 1805 1806 1807 1808 1809 1810 1811 1812 1813 1814 1815
1816 1817 1818 1819 1820 1821 1822 1823 1824 1825 1826 1827 1828 1829 1830 1831
1832 1833 1834 1835 1836 1837 1838 1839 1840 1841 1842 1843 1844 1845 1846 1847
1848 1849 1850 1851 1852 1853 1854 1855 1856 1857 1858 1859 1860 1861 1862 1863
1864 1865 1866 1867 1868 1869 1870 1871 1872 1873 1874 1875 1876 1877 1878 1879
1880 1881 1882 1883 1884 1885 1886 1887 1888 1889 1890 1891 1892 1893 1894 1895
1896 1897 1898 1899 1900 1901 1902 1903 1904 1905 1906 1907 1908 1909 1910 1911
1912 1913 1914 1915 1916 1917 1918 1919 1920 1921 1922 1923 1924 1925 1926 1927
1928 1929 1930 1931 1932 1933 1934 1935 1936 1937 1938 1939 1940 1941 1942 1943
1944 1945 1946 1947 1948 1949 1950 1951 1952 1953 1954 1955 1956 1957 1958 1959
1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975
1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991
1992 1993 1994 1995 1996 1997 1998 1999 2000
```

[1]:
```python
stu_per = 85.47
if stu_per >= 90:
    print('O')
elif stu_per >= 80:
    print('A')
elif stu_per >= 70:
    print('B')
elif stu_per >= 60:
    print('C')
elif stu_per >= 50:
    print('D')
else:
    print('E')
stu2_per = 47.86
# per >= 90 --> O
# per b/w 80 and 90 --> A
# per b/w 70 and 80 --> B
# per b/w 60 and 70 --> C
# per b/w 50 and 60 --> D
# per <50 --> E
```

```
A
```

[5]:
```python
# Palindromes in a range using functions (with return type)
def reverse(num): # num = 121
    rev = 0
    while num > 0:
        r = num % 10
        rev = rev * 10 + r
        num = num // 10
    return rev
a, b = map(int, input().split())
for i in range(a, b + 1):
    if i == reverse(i):
        print(i, end = ' ')
```

```
1000 2000
1001 1111 1221 1331 1441 1551 1661 1771 1881 1991
```

[6]:
```
def fun(a, b):
    return 10
    return a + b
    return a - b
print(fun(10, 20))
```

```
10
```

[7]:
```
def check(a):
    if a > 10:
        return 1
    elif a < 10:
        return 2
print(check(11))
```

```
1
```

[9]:
```
# primes in a range
for i in range(10, 20):
    is_prime = True
    for j in range(2, int(i ** 0.5) + 1):
        if i % j == 0:
            is_prime = False
            break
    if is_prime == True:
        print(i, end = ' ')
```

```
11 13 17 19
```

[18]:
```
# Primes in range using functions
def is_prime(n):
    if n == 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
a, b = map(int, input().split())
for i in range(a, b + 1):
    if is_prime(i) == True:
        print(i, end = ' ')
```

```
100 200
101 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197
199
```

[21]:
```
# next prime
def is_prime(n):
    if n == 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
z = int(input())
np = z + 1
while True:
    if is_prime(np) == True:
        print(np)
        break
    np += 1
```

```
11
13
```

1.2 type conventions in user defined functions

Type conventions are used to convey the details about a function to user like - What type of arguments should she/he pass while calling the function - What will be the return type of function Syntax:

```
def function_name(parameter1: type, parameter2: type, ...) -> returntype:
    function_body
```

[3]:
```
def add(a: int, b: int, c: int) -> None:
    print(a + b + c)

add(10, 20, 30)
```

```
60
```

[6]:
```
# That takes a name and prints the name for n times
def fun(x: str, n: int) -> None:
    for i in range(n):
        print(x)
```

```
fun('pavan', 10)
```

```
pavan
pavan
pavan
pavan
pavan
pavan
pavan
pavan
pavan
pavan
```

[7]:
```
def Add(a: int, b: int) -> int:
    return a + b
Add(10, 20)
```

[7]:
```
30
```

[8]:
```
def Add(num1: int, num2: int) -> int:
    return num1 + num2

print(Add(10, 20))
```

```
30
```

[71]:
```
def reverse(num: int) -> int:
    rev = 0
    while num > 0:
        r = num % 10
        rev = rev * 10 + r
        num = num // 10
    return rev

print(reverse(123))
```

```
321
```

1.3 Doc strings in functions

- Doc strings are used to describe the functionality of a function in words
- Doc strings should be written as the very first line(s) of the function in either
  - Triple Quotes (" ""Doc string here"" ") if multiple lines are present in Doc string
  - or Single ('Doc string here') or Double ("Doc string here") quotes if only single line is present as Doc string
- Doc string will be fetched by anyone who wants to get some help on the function that you created
- To the get the help on any function (built_in or user defined) one can do
  - help(function_name) or

– print(function_name.*doc*)

[9]:
```python
help(max) # built_in function
```

```
Help on built-in function max in module builtins:

max(…)
    max(iterable, *[, default=obj, key=func]) -> value
    max(arg1, arg2, *args, *[, key=func]) -> value

    With a single iterable argument, return its biggest item. The
    default keyword-only argument specifies an object to return if
    the provided iterable is empty.
    With two or more arguments, return the largest argument.
```

[12]:
```python
def myfunction(a, b, c):
    """Description:
    Takes: Three integers a, b and c
    Prints: a multiplied with b and added to c"""
    print(a * b + c)
myfunction(1, 2, 3)
# help(myfunction)
print(myfunction.__doc__)
```

```
5
Description:
    Takes: Three integers a, b and c
    Prints: a multiplied with b and added to c
```

[15]:
```python
def check(a):
    '''Takes a number
    prints number + 1'''
    print(a + 1)

print(check.__doc__)
```

```
Takes a number
    prints number + 1
```

[19]:
```python
def print_all(a, b):
    'Takes a and b prints all numbers from a to b (inclusive)'
    for i in range(a, b + 1):
        print(i, end = ' ')
# print_all(10, 20)
print(print_all.__doc__)
```

```
Takes a and b prints all numbers from a to b (inclusive)
```

1.4 positional and default arguments

[20]:
```
def location(city, country, continent):
    print(f'{city} is in {country} which is located in {continent}')

location('Mumbai', 'India', 'Asia') # positional arguments
```

```
Mumbai is in India which is located in Asia
```

[21]:
```
def location(city, country, continent):
    print(f'{city} is in {country} which is located in {continent}')

location('India', 'Mumbai', 'Asia') # positional arguments
```

```
India is in Mumbai which is located in Asia
```

[24]:
```
def location(city: str, country: str, continent: str) -> None:
    print(f'{city} is in {country} which is located in {continent}')

location(country='India', city='Mumbai', continent='Asia') # calling using␣
 ↪keywords
```

```
Mumbai is in India which is located in Asia
```

[26]:
```
def location(city, country, continent):
    print(f'{city} is in {country} which is located in {continent}')

location('Delhi', continent='Asia', country='India') # calling using keywords
```

```
Delhi is in India which is located in Asia
```

1.5 Default values to parameters in functions

[27]:
```
def location(city, country, continent):
    print(f'{city} is in {country} which is located in {continent}')

location('Mumbai', 'India', 'Asia') # valid?
```

```
Mumbai is in India which is located in Asia
```

[28]:
```
def location(city, country, continent):
    print(f'{city} is in {country} which is located in {continent}')

location('India', 'Asia', 'Mumbai') # valid?
```

```
India is in Asia which is located in Mumbai
```

[29]:
```
def location(city, country, continent):
    print(f'{city} is in {country} which is located in {continent}')

location('India', 'Asia') # valid?
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [29], in <cell line: 4>()
      1 def location(city, country, continent):
      2     print(f'{city} is in {country} which is located in {continent}')
----> 4 location('India', 'Asia')

TypeError: location() missing 1 required positional argument: 'continent'
```

[30]:
```
def location(city, country, continent):
    print(f'{city} is in {country} which is located in {continent}')

location('India') # valid?
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [30], in <cell line: 4>()
      1 def location(city, country, continent):
      2     print(f'{city} is in {country} which is located in {continent}')
----> 4 location('India')

TypeError: location() missing 2 required positional arguments: 'country' and␣
↪'continent'
```

[31]:
```
def location(city, country, continent):
    print(f'{city} is in {country} which is located in {continent}')

location() # valid?
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [31], in <cell line: 4>()
      1 def location(city, country, continent):
      2     print(f'{city} is in {country} which is located in {continent}')
----> 4 location()

TypeError: location() missing 3 required positional arguments: 'city',␣
↪'country', and 'continent'
```

[32]:
```
# default values to parameteres
def location(city, country, continent='Asia'): # default value
    print(f'{city} is in {country} which is located in {continent}')
```

```
location('Mumbai', 'India') # valid? YES
```

```
Mumbai is in India which is located in Asia
```

[34]:
```
# default values to parameteres
def location(city, country, continent='Asia'): # default value
    print(f'{city} is in {country} which is located in {continent}')

location('Berlin', 'Germany', 'Europe') # valid? YES
```

```
Berlin is in Germany which is located in Europe
```

[35]:
```
# default values to parameteres
def location(city, country, continent='Asia'): # default value
    print(f'{city} is in {country} which is located in {continent}')

location('Berlin') # valid? YES
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [35], in <cell line: 5>()
      2 def location(city, country, continent='Asia'): # default value
      3     print(f'{city} is in {country} which is located in {continent}')
----> 5 location('Berlin')

TypeError: location() missing 1 required positional argument: 'country'
```

[36]:
```
# default values to parameteres
def location(city, country='India', continent='Asia'): # default value
    print(f'{city} is in {country} which is located in {continent}')

location('Delhi') # valid? YES
```

```
Delhi is in India which is located in Asia
```

[37]:
```
# default values to parameteres
def location(city, country='India', continent='Asia'): # default value
    print(f'{city} is in {country} which is located in {continent}')

location() # valid? YES
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [37], in <cell line: 5>()
      2 def location(city, country='India', continent='Asia'): # default value
      3     print(f'{city} is in {country} which is located in {continent}')
----> 5 location()
```

```
TypeError: location() missing 1 required positional argument: 'city'
```

[39]:
```
# default values to parameteres
def location(city, country='India', continent='Asia'): # default value
    print(f'{city} is in {country} which is located in {continent}')

location('Tokyo', 'Japan') # valid? YES
```

```
Tokyo is in Japan which is located in Asia
```

[42]:
```
# default values to parameters
def location(city, country='India', continent='Asia'): # default value
    print(f'{city} is in {country} which is located in {continent}')

location('Harare', 'Zimbabwe', 'Africa') # valid? YES
```

```
Harare is in Zimbabwe which is located in Africa
```

[46]:
```
# default values to parameters
def location(city='Mumbai', country='India', continent='Asia'): # default value
    print(f'{city} is in {country} which is located in {continent}')

# location() # valid?
# location('Delhi') # valid?
# location('Shanghai', 'China') # valid?
location('New York', 'US', 'North America') # valid?
```

```
New York is in US which is located in North America
```

[56]:
```
lst = [10, 20, 30]
print(sum(lst, 10))
```

```
70
```

[61]:
```
print(10, 20, 30, 40, sep='pavan') # space
```

```
10pavan20pavan30pavan40
```

1.6 Functions without arguments with return type

[66]:
```
def gi():
    return int(input()) # 10

def gf():
    return float(input()) # 22.5

def get_mul_int():
    return map(int, input().split())
```

```
a, b, c = get_mul_int()
print(a + b + c)
p, q, r = get_mul_int()
print(p + q + r)
```

```
10 20 30
60
40 50 60
150
```

[70]:
```
# Multipurpose read function
def read(datatype, mul=False):
    if mul == False:
        return datatype(input())
    else:
        return map(datatype, input().split())
# a = read(int)
# print(a)
# b = read(float)
# print(b)
# c = read(str)
# print(c)
a, b = read(int, True)
print(a + b)
```

```
10 20
30
```

[ ]:
```
a = int(input())
b = float(input())
c = str(input())
x, y = map(int, input().split())
p, q = map(float, input().split())
m, n = map(str, input().split())
```

1.6.1 Multipurpose read function

[ ]:
```
def read(datatype, mul=False):
    if mul == False:
        return datatype(input())
    else:
        return map(datatype, input().split())

a, b, c = read(int, True)
```

1.7 Arbitrary number of arguments to function

[11]:
```
print(max(10, 20))
```

```
20
```

[10]:
```
def my_max(a, b):
    if a > b:
        return a
    else:
        return b

print(my_max(10, 20, 30))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [10], in <cell line: 7>()
      4     else:
      5         return b
----> 7 print(my_max(10, 20, 30))

TypeError: my_max() takes 2 positional arguments but 3 were given
```

[21]:
```
def my_sum(*nums): # nums = (10, 20, 30, 40, 50)
    s = 0
    for i in nums:
        s += i
    return s

print(my_sum(10, 20))
print(my_sum(10, 20, 30))
print(my_sum(10, 20, 30, 40))
print(my_sum(10, 20, 30, 40, 50))
```

```
30
60
100
150
```

[22]:
```
def my_sum(*nums): # nums = (10, 20, 30, 40, 50)
    s = 0
#     print(type(nums))
    for i in nums:
        s += i
    return s

print(my_sum(10, 20))
print(my_sum(10, 20, 30))
print(my_sum(10, 20, 30, 40))
print(my_sum(10, 20, 30, 40, 50))
```

```
<class 'tuple'>
30
<class 'tuple'>
60
<class 'tuple'>
100
<class 'tuple'>
150
```

[ ]:
```
print(prod(1, 2)) # 2
print(prod(1, 2, 3)) # 6
print(prod(1, 2, 3, 4)) # 24
```

[ ]:
```

```

[31]:
```
a, b, c = 10, 20, 30
print(a, b, c)
```

```
10 20 30
```

[32]:
```
a, b, c = 10, 20
print(a, b, c)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [32], in <cell line: 1>()
----> 1 a, b, c = 10, 20
      2 print(a, b, c)

ValueError: not enough values to unpack (expected 3, got 2)
```

[33]:
```
a, b, c = [10, 20, 30]
print(a, b, c)
```

```
10 20 30
```

[34]:
```
a, b, c = [10, 20, 30, 40]
print(a, b, c)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [34], in <cell line: 1>()
----> 1 a, b, c = [10, 20, 30, 40]
      2 print(a, b, c)
```

```
ValueError: too many values to unpack (expected 3)
```

[35]:
```
*x, a = 10, 20, 30
print(x) # list
print(a)
```

```
[10, 20]
30
```

[37]:
```
a, b, *x = 10, 20, 30, 40, 50, 60
print(x) # list
print(a)
print(b)
```

```
[30, 40, 50, 60]
10
20
```

[38]:
```
a, *b, x = 10, 20, 30, 40, 50, 60
print(x) # list
print(a)
print(b)
```

```
60
10
[20, 30, 40, 50]
```

[39]:
```
*a, b, x = 10, 20, 30, 40, 50, 60
print(x) # list
print(a)
print(b)
```

```
60
[10, 20, 30, 40]
50
```

[ ]:
```

```

[46]:
```
a, b = map(int, input().split())
```

```
10 20 30
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [46], in <cell line: 1>()
----> 1 a, b = map(int, input().split())
```

```
ValueError: too many values to unpack (expected 2)
```

```
[41]: a, b = 10, 20, 30
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [41], in <cell line: 1>()
----> 1 a, b = 10, 20, 30

ValueError: too many values to unpack (expected 2)
```

```
[47]: a, *b = map(int, input().split())
      print(a)
      print(b)
```

```
10 20 30 40 50
10
[20, 30, 40, 50]
```

```
[48]: print(*b)
```

```
20 30 40 50
```