Chapter 5: Lists

Last Update: March 15, 2023

Program 1: A comparison of various data structures.

```
05 10-data structure.py
# a list
languages = ["Python", "Java", "Fortran"]
# a tuple
History = ("Python", 1991)
# a set
Vowel = {"a", "e", "i", "o", "u"}
print({"a", "b"}=={"b", "a"})
# a dictionary
Capital = {"USA": "Washington, DC",
           "Italy": "Rome",
           "Ukraine": "Kyiv",
           "England": "London"}
# More
Languages_Year = [("Python", 1991), ("C++", 1983)]
```

Program 2: This example shows multiple ways to initialize a list. List elements may be an expression, so you should evaluate the expression first.

05 11-creating a list.py

```python
def next():
    input("\nNext> ")

# List initialization
a = [1, 2, 3, 4]
print(a)
next()

b = []
print(b)
next()

c = ['banana', 'apple', 'pear'] + ['watermelon']
print(c)
next()

d = ["apple", 20, 3.5]
print(d)
next()

e = [[1,2], [3,4,5], 99]
print(e)
next()

f = [a, c]
print(f)

# Accessing List element
print(f[1][2])
```

Program 3: More list examples with expressions.

05 12-list.py

```python
x = 100
y = "UH"
list1 = [2**3, x/3, y*2]
print(list1)
```

Program 4: This example shows how to initialize and change a list. In each of the three examples, an empty list is initialized first. Then we change the list by adding an element using three different methods. In all three cases, the list is changed to what we intended. However, the list's identity (or memory address) may vary depending on how the list is altered. Does it matter? See the following example for the answer.

```
05  13-change.py
def next():
    input("\nNext> ")

a = []
print("*** Case 1: Concatenation")
print(f"\tBefore: {id(a)}, {a}")
a = a+[1]
print(f"\tAfter:  {id(a)}, {a}")
next()

a = []
print("*** Case 2: Assignment Operator")
print(f"\tBefore: {id(a)}, {a}")
a += [2]
print(f"\tAfter:  {id(a)}, {a}")
next()

a = []
print("*** Case 3: List Append")
print(f"\tBefore: {id(a)}, {a}")
a.append(3)
print(f"\tAfter:  {id(a)}, {a}")
```

Program 5: (Continue from the last program) In this example, we move the list assignment code into a function. One of the three cases did not work as we expected. We added a fourth case to correct the one that did not work. What did you learn from these examples?

In Case 1, we created a new list (using the old list) and assigned it to the variable. In Cases 2 and 3, we made the change in situ.

05 14-list parameter.py

```python
def next():
    input("\nNext> ")

def list_para1(b):
    b = b+[1]
    return b

def list_para2(b):
    b += [2]
    return b

def list_para3(b):
    b.append(3)
    return b

a = []
print("*** Case 1: Concatenation")
print(f"\tBefore: {id(a)}, {a}")
list_para1(a)
print(f"\tAfter:  {id(a)}, {a}")
next()

a = []
print("*** Case 2: Assignment Op")
print(f"\tBefore: {id(a)}, {a}")
list_para2(a)
print(f"\tAfter:  {id(a)}, {a}")
next()

a = []
print("*** Case 3: List Append")
print(f"\tBefore: {id(a)}, {a}")
list_para3(a)
print(f"\tAfter:  {id(a)}, {a}")
next()

a = []
print("*** Case 4: Concatenation")
print(f"\tBefore: {id(a)}, {a}")
a = list_para1(a)
print(f"\tAfter:  {id(a)}, {a}")
```

Program 6: This example shows lists are ordered, indexed, mutable structures.

05 15-list order.py

```python
def next():
    input("\nNext> ")

languages = ["Python", "Java", "C++", "Fortran"]
# Ordered
print(languages)
next()

# Indexed
for idx in range(len(languages)):
    print(idx, languages[idx])
next()

# Duplicate allowed
languages = ["Python", "Java", "C++", "Fortran", "Python"]
print(languages)
next()

fruits = ["apple", "banana", "cherry"]
print(fruits)
# Change item
fruits[1] = "pear"
print(fruits)
next()

# Add or remove item
fruits.append("banana")
print(fruits)
```

Program 7: Initializing and accessing lists.

```
05 21-def.py
def next():
    input("\nNext> ")

odd = [1, 3, 5, 7, 9]
print(f"{odd} in a list form.")
next()
print(f"odd[2] = {odd[2]}")
next()

idx = int(input("Pick an index: "))
print(f"odd[{idx}] = {odd[idx]}")
next()

print("For loop + List")
for elem in odd:
    print(elem, end=', ')
print()
next()

for i, elem in enumerate(odd):
    print(elem, end='\n' if i==len(odd)-1 else ', ')
```

Program 8: This example demonstrates that lists are mutable, i.e., once a list has been created, (1) elements can be modified, and (2) individual elements can be replaced. To compare, an integer or a string is not mutable. If we change an integer value, its id changes, but if we change a list, its id remains the same.

```
05 22-mutable.py
def next():
    input("\nNext> ")

numbers = [1, 3, 5, 7, 9]
print(numbers)
next()
# Access an element with an index
idx = int(input("Enter an index: "))
if 0<=idx<len(numbers):
    print(f"  numbers[{idx}] = {numbers[idx]}")
else:
    print(f"{idx} is outside the list range")
next()

# Changing an element of the list
idx = int(input("\nPick an index of the list: "))
value = int(input("Enter a new value: "))
numbers[idx] = value
print(f"  numbers[{idx}] = {numbers[idx]}")
next()

print(numbers)
for num in numbers:
    num += 1
    print(num)
print(numbers)
next()
```

Program 9: List elements can be strings too.

```
05 23-list of str.py
list1 = ['kiwi', 'apple', 'watermelon']
print(list1)

list1[0] = 'banana'
print(list1)

list1[2] = 99
print(list1)
```

Program 10: The operator “in” checks for membership of an element in a list.

```
05  24-in as op.py
```

```python
fruits = ['banana', 'apple', 'mango', 'pear']

print('pear' in fruits)
print('watermelon' in fruits)
```

Program 11: Negative index.

```
05  25-neg index.py
```

```python
def next():
    input("> ")

fruits = ['banana', 'apple', 'mango', 'pear']

print("Using positive index")
for i in range(len(fruits)):
    print(i, fruits[i])
next()

print("Using negative index")
for i in range(-1, -len(fruits)-1, -1):
    print(i, fruits[i])
next()

print("Both positive and negative indices")
for i in range(len(fruits)):
    print(i, i-len(fruits), fruits[i])
```

Program 12: Example of using enumerate().

```
05 31-enumerate.py
```

```python
def next():
    input("\nNext> ")

fruits = ['apple', 'banana', 'mango', 'pear', 'watermelon']

i = 0
while i < len(fruits):
    print(f'{i}: {fruits[i]}')
    i += 1
next()

for i in range(len(fruits)):
    print(f'{i}: {fruits[i]}')
next()

for f in fruits:
    print(f)
next()

for i, f in enumerate(fruits):
    print(f'{i}: {f}')
next()

for i, f in enumerate(fruits, 1):
    print(f'{i}: {f}')
print()
```

Program 13: Now, we can provide a solution to Program 7 above.

05 32-enumerate.py

```python
def next():
    input("\nNext> ")

numbers = [1, 3, 5, 7, 9]
print(numbers, id(numbers))
next()

print(numbers, id(numbers))
for i, num in enumerate(numbers):
    num += 1
    print(num)
print(numbers, id(numbers))
next()

print(numbers, id(numbers))
for i, num in enumerate(numbers):
    numbers[i] += 1
    print(num)
print(numbers, id(numbers))
next()

print(f"{numbers[2]:3d} {id(numbers[2])}")
numbers[2] = -9
print(f"{numbers[2]:3d} {id(numbers[2])}")
next()

a = b = [5, 4, 3, 2, 1]
print(f"{a} {id(a)}")
a[1] = '???'
print(f"{b} {id(b)}")
```

Program 14: More on list id. Multiple assignments of a list will assign the same id to the lists. On the other hand, if we initialize two variables to the same list separately, their identities are different.

05 33-list id.py

```python
def next():
    input("\nNext> ")

a = [1, 3, 5, 7, 9]
print(a, id(a))
next()

b = [1, 3, 5, 7, 9]
print(b, id(b))
next()

a = b = [1, 3, 5, 7, 9]
print(id(a), id(b))
```

Program 15:

```
05 40-print.py
def next():
    input("\nNext> ")

prime_list = [2, 3, 5, 7, 11, 13, 17, 19]
print(prime_list)
next()

print("*** Case 1: iterator")
for p in prime_list:
    print(f"\t\t{p}")
next()

print("*** Case 2: index")
for i in range(len(prime_list)):
    print(f"\t{i}: {prime_list[i]}")
next()

print("*** Case 3: enumerate")
for i, p in enumerate(prime_list, 1):
    print(f"\t{i}: {p}")
```

Program 16:

```
05 41-operaor.py
def next():
    input("> ")

a = [1, 2, 3]
b = [4, 5, 6]

print("*** Case 1: A list of multiple types.")
x = ['Stephen', '713-743-3335', 66, True]
print(f"\t{x}")
next()

print("*** Case 2: Concatenation of lists.")
c = a + b
print(f"\t{c}")
print(f"\t{b*3}")
next()

print("*** Case 3: Concatenation.")
d = []
for i in range(10):
    d = d+[i]
    print(f'\t{i}: {d}')
next()

print("*** Case 4: Search")
test = int(input("\tEnter a number: "))
if test in d:
    print(f"\t{test} is in {d}")
else:
    print(f"\t{test} is not in {d}")
```

Program 17:

```
05  42-print.py
```

```python
def next():
    input("\nNext> ")

def max1(a):
    max, index = a[0], 0
    for i in range(1,len(a)):
        if a[i] > max:     # try > or >=
            max = a[i]
            index = i
    return index             # may also return the max

def max2(b):
    max = b[0]
    for x in b:
        if x>max:
            max = x
    return max

def max3(b):
    max, idx = b[0], 0
    for i, x in enumerate(b):
        if x>max:
            max = x
            idx = i
    return idx

a = [2, 3, 25, 4, 9, 8, 7, 16, 25]
print("*** Case 1:")
print(f"\tMax 1: {max1(a)} -> {a[max1(a)]}")
next()

print("*** Case 2:")
print(f"\tMax 2: X    {max2(a)}")
next()

print("*** Case 3:")
print(f"\tMax 3: {max3(a)} -> {a[max3(a)]}")
next()

print("*** Case 4:")
b = [2, 3, 25, 4, 9, 8, 7, 16, 25]
print(f"\tUsing built-in max() function: {max(b)}")
```

Program 18:

```
05 43-print.py
def next():
    input("\nNext> ")

def init_array(a, size):
    for i in range(size):
        a += [0]
    return a  # must return the list

def print_array(a):
    for i in range(len(a)):
        print(f"\t{a[i]}", end=', ')
    print()

print("*** Case 1")
a = []
init_array(a, 10)
print_array(a)
next()

print("*** Case 2")
a = []
a = init_array(a, 7)
print_array(a)
next()

print("*** Case 3")
a = []
print(f"\t{id(a)}, {a}")
a.append(1)
print(f"\t{id(a)}, {a}")
a.append(2)
print(f"\t{id(a)}, {a}")
```

Program 19:

```
05 51-insert.py
list = list(range(7))
print(list)

print("\n*** Case 1: insert 99 at -1")
list.insert(-1, 99)
print(list)

print("\n*** Case 2: insert 88 at 8")
list.insert(8, 88)
print(list)

print("\n*** Case 3: insert 77 at 20")
list.insert(20, 77)
print(list)
# No position 20!
```

Program 20:

```
05 54-neg index.py
```

```python
def print_list(mylist):
    for elem in mylist:
        print(f'{elem:4d}', end=' ')
    print()

pos_index = list(range(0,10,1))
print_list(pos_index)

# list comprehension
mylist = [100+elem*11 for elem in pos_index]
print_list(mylist)

neg_index = list(range(-10,0,1))
print_list(neg_index)
print()

print(f'[ 0:10: 1] -> {mylist[ 0:10: 1]}')
print(f'[-1:-4:-1] -> {mylist[-1:-4:-1]}')
print(f'[ 5:-1: 1] -> {mylist[ 5:-1: 1]}')
print(f'[ 5: 0:-1] -> {mylist[ 5: 0:-1]}')
```

Program 21:

```
05 55-default.py
```

```python
a = list(range(10))
print(a)
print(a[:])
print(a[3:7])
print(a[3:])
print(a[:7])
print()
print(a[0:-1])
print(a[3:-2])
print(a[-1:7])
print(a[-1:7:-1])
```

Program 22:

```
05 56-equivalent.py
a = "banana"
b = "banana"
c = b
print('string', id(a), id(b), id(c))

a = [1, 2, 3]
b = [1, 2, 3]
c = b
print('list  ', id(a), id(b), id(c))

a = 99
b = a*1
print('      99', id(a), id(b))

a = 99999999
b = a*1
print('99999999', id(a), id(b))

a = 99999999
b = a
print('99999999', id(a), id(b))
```

Program 23:

```
05 61-methods.py
```

```python
def next():
    input("\nNext> ")

list1 = list(range(10, 20))
print('Original List: ', list1)
next()

list1.append(99)
print('Append 99 to list 1:', list1)
list2 = list1.copy()
next()

print("list2: ", list2)
list2.append(15)
print('Append 15 to list2: ', list2)
next()

print('Count 15 in List2: ', list2.count(15))
next()

list2.clear()
print('Clear List2: ', list2)
print('Clear List1: ', list1)
next()

list3 = [100, 101, 102, 103, 104]
list1.extend(list3[1:3])
print('Extend a slice to List1: ', list1)
next()

list3 = [100, 101, 102, 103, 104]
list1.append(list3[1:3])
print('Append a slice to List1: ', list1)
next()

print('Index of 101 in [8,13) range: ', list1.index(101, 8, 13))
next()

list1.insert(5, -1)
print('Insert -1 at position 5: ', list1)
next()

list1.remove(101)
print('Remove 101 from List1: ', list1)
next()

list1.insert(1, 99)
print('Insert 99 at position 1', list1)
```

Program 24: This program does not work. Why? How to fix it?

```
05 62a-pop.py
my_list = list(range(100, 109))
print(my_list)

for i in range(len(my_list)): # Does not re-evaluate the length
    print("* ", i)
    if my_list[i]==103:
        my_list.pop(i)
        print("Pop 103")
    else:
        my_list[i] += 100

print(my_list)
```

Program 25: This is the correct solution for the last program.

```
05 62b-pop.py
my_list = list(range(100, 110))
print(my_list)
i = 0
while i < len(my_list):
    print("* ", i)
    if my_list[i]==103:
        my_list.pop(i)
        print("Pop 103")
    else:
        my_list[i] += 100
    i += 1
print()
print(my_list)
```

Program 26: Two additional solutions. One of them does not work.

```
05 62c-pop.py
```

```python
def next():
    input("\nNext> ")

my_list = list(range(100, 110))
print(my_list)
next()

for i, v in enumerate(my_list):
    if v == 103:
        my_list.pop(i)
    else:
        my_list[i] += 100
print()
print(my_list)
next()

my_list = list(range(100, 110))
i = 0
while i < len(my_list):
    if my_list[i]==103:
        my_list.pop(i)
    else:
        my_list[i] += 100
        i += 1
print()
print(my_list)
```

Program 27:

```
05 63-copy.py
```

```python
def next():
    input("\nNext> ")

old = list(range(10))
print(f"\t{id(old)}, old list = {old}")
new = old
print(f"\t{id(new)}, new list = {new}")

# Now we take an element out from the new list
print(f"\n*** Remove {new.pop()} from the new list")
next()

print(f"\t{id(new)}, new list = {new}")
next()

print(f"\t{id(old)}, old list = {old}")
```

Program 28:

05 64a-shallow copy.py

```python
def next():
    input("> ")

fruit1 = ['apple', 'banana', 'cherry']

fruit2 = fruit1
print("*** Case 1: Alias")
print(f"\tfruit1 ID = {id(fruit1)} {fruit1}" )
print(f"\tfruits ID = {id(fruit2)} {fruit2}" )
next()

print("*** Case 2: Shallow Copy")
fruit2 = fruit1.copy()
print(f"\tfruit1 ID = {id(fruit1)} {fruit1}" )
print(f"\tfruit2 ID = {id(fruit2)} {fruit2}" )
next()

print("*** Case 3: Sharing elements (Shallow)")
print(f"\tfruit1[0] ID = {id(fruit1[0])} {fruit1[0]}" )
print(f"\tfruit2[0] ID = {id(fruit2[0])} {fruit2[0]}" )
next()

print("*** Case 4: Changing an element")
fruit1[0] = 'pear'
print(f"\tfruit1[0] ID = {id(fruit1[0])} {fruit1[0]}" )
print(f"\tfruit2[0] ID = {id(fruit2[0])} {fruit2[0]}" )
```

Program 29:

05 64b-deep copy.py

```python
import copy
def next():
    input("> ")

fruit1 = ['apple', 'banana', 'cherry']

print("*** Case 1: Deep Copy")
fruit2 = copy.deepcopy(fruit1)
print(f"\tfruit1 ID = {id(fruit1)} {fruit1}" )
print(f"\tfruit2 ID = {id(fruit2)} {fruit2}" )
next()

print("*** Case 2: Sharing elements")
print(f"\tfruit1[0] ID = {id(fruit1[0])} {fruit1[0]}" )
print(f"\tfruit2[0] ID = {id(fruit2[0])} {fruit2[0]}" )
next()

print("*** Case 3: Changing an element")
fruit1[0] = 'pear'
print(f"\tfruit1[0] ID = {id(fruit1[0])} {fruit1[0]}" )
print(f"\tfruit2[0] ID = {id(fruit2[0])} {fruit2[0]}" )
```

Program 30:

```
05  65-index.py
```

```python
def next():
    input("\nNext> ")

def search(x,alist):
    for i, v in enumerate(alist):
        if v == x:
            return i
    return None

a = [1, 2, 5, 9, 7, 9, 2]
print(f"\ta = {a}")
next()

print("*** Search")
print(f"\tSearching for 2: {search(2, a)}")
print(f"\tSearching for 9: {search(9, a)}")
print(f"\tSearching for 4: {search(4, a)}")
next()

print("*** index()")
print(f"\tSearching for 2: {a.index(2)}")
print(f"\tSearching for 9: {a.index(9)}")
print(f"\tSearching for 4: {a.index(4)}")
```

Program 31:

```
05 66-remove.py
```

```
def next():
    input("\nNext> ")

print("*** Case 1: Remove 3")
list = [1, 2, 3, 9, 3, 5, 2, 8]
print("The list:", list)
for elem in list:
    if elem==3:
        list.remove(3)
        break
    else:
        print(elem, end=', ')
print()
print("The list:", list)
next()

print("\n*** Case 2: Remove 2")
list = [1, 2, 3, 9, 3, 5, 2, 8]
print(list)
i = 0
while i <len(list):
    if list[i]==2:
        list.remove(2)
        break
    else:
        print(list[i], end=', ')
    i += 1
print()
print(list)
next()

print("\n*** Case 3: Remove 5")
list = [1, 2, 3, 9, 3, 5, 2, 8]
print(list)
i = 0
while i <len(list):
    if list[i]==5:
        list.remove(5)
        break
    else:
        print(list[i], end=', ')
        i += 1
print()
print(list)
```

Program 32

```
05 67-reverse.py
```

```python
def next():
    input("\nNext> ")

fruits = ['apple', 'banana', 'cherry']

print(f"*** Reverse {fruits}")
fruits.reverse()
print(fruits)
next()

print(f"*** Sort {fruits}")
fruits.sort()
print(fruits)
next()

print(f"*** insert, index, append {fruits}")
fruits.insert(1,"mango")
print(fruits.index("banana"))
fruits.append("watermelon")
print(fruits)
next()

print(f"*** pop {fruits}")
fruits.pop(2)
print(fruits)
```

Program 33:

```
05 71-2d list.py
```

```python
def printlist(list):
    for i in range(len(list)):
        for j in range(len(list[i])):
            print(f"{list[i][j]:5d}", end=' ')
        print()
    print()

list = create(6, 4)
printlist(list)

list1 = [
    [1],
    [2, 3],
    [4, 5, 6]
]
print(list1)
```

Program 34:

```
05 72-2d list.py
def create(m):
    list = []
    for i in range(m):
        sublist=[]
        for j in range(i+1):
            sublist.append(100*(i+1)+j)
        list.append(sublist)
    return list

def printlist(list):
    for sublist in list:
        for elem in sublist:
            print(f"{elem:5d}", end=' ')
        print()
    print()

list = create(3)
printlist(list)
print(list)
```

Program 35:

```
05 73-2d max.py
mylist = [
    [10, 20, 30],
    [23, 20, 26],
    [15, 39, 23]
]

# Case 1
max = 0
for i in range(len(mylist)):
    for j in range(len(mylist[i])):
        if mylist[i][j]>max:
            max = mylist[i][j]
print(max)

# Case 2
max = 0
for sub in mylist:
    for elem in sub:
        if elem>max:
            max = elem
print(max)
# Which one is better?
```

Program 36:

```
05    74-2d transpose.py
```

```python
mylist = [
    [10, 20, 30],
    [23, 20, 26],
    [-1, -2, -3]
]

for i in range(len(mylist)):
    for j in range(len(mylist[i])):
        print(mylist[i][j], end=' ')
    print()
print()
for i in range(len(mylist)):
    for j in range(len(mylist[i])):
        print(mylist[j][i], end=' ')
    print()
```

Program 37:

```
05    81-list comprehension.py
```

```python
def print_list(mylist):
    for elem in mylist:
        print(f'{elem:4d}', end=' ')
    print()

pos_index = list(range(0,10,1))
print_list(pos_index)

# list comprehension
mylist = [elem*11 for elem in pos_index]
print_list(mylist)

mylist = [elem**2 for elem in pos_index]
print_list(mylist)
```

Program 38:

```
05 82-list comprehension.py
base_list = list(range(16))
print(f"Base list = {base_list}")

new_list = []
for n in base_list:
    if n%3 == 0:
        new_list.append(n/2)
print(f"    New List  = {new_list}")
print(f"    Base list = {base_list} unchanged")

print("\nUsing list comprehension in one line")

new_list = [n/2 for n in base_list if n%3==0]
print(f"    New List  = {new_list}")
print(f"    Base list = {base_list} unchanged")

# There is no reason to save space. We could do the following:
new_list = [
    n/2                    # computation/modification
    for n in base_list     # the base list
    if n%3==0              # selection/filtering
]
# if that helps
print("\nThe same result using 5 lines of code")
print(f"    New List  = {new_list}")
print(f"    Base list = {base_list} unchanged")
```

Program 39:

```
05 83-init.py
import random as r

def printsq(arr):
    size = len(arr)
    for i in range(size):
        for j in range(size):
            print(f"{arr[i][j]:3d}", end = ' ')
        print()
    print()

size = 6
array6x6 = [[0 for j in range(size)] for i in range(size)]
printsq(array6x6)

array6x6 = [[1 if i==j else 0 for j in range(size)] for i in
range(size)]
printsq(array6x6)

size = 4
array4x4 = [[r.randrange(0,99) for j in range(size)] for i in
range(size)]
```