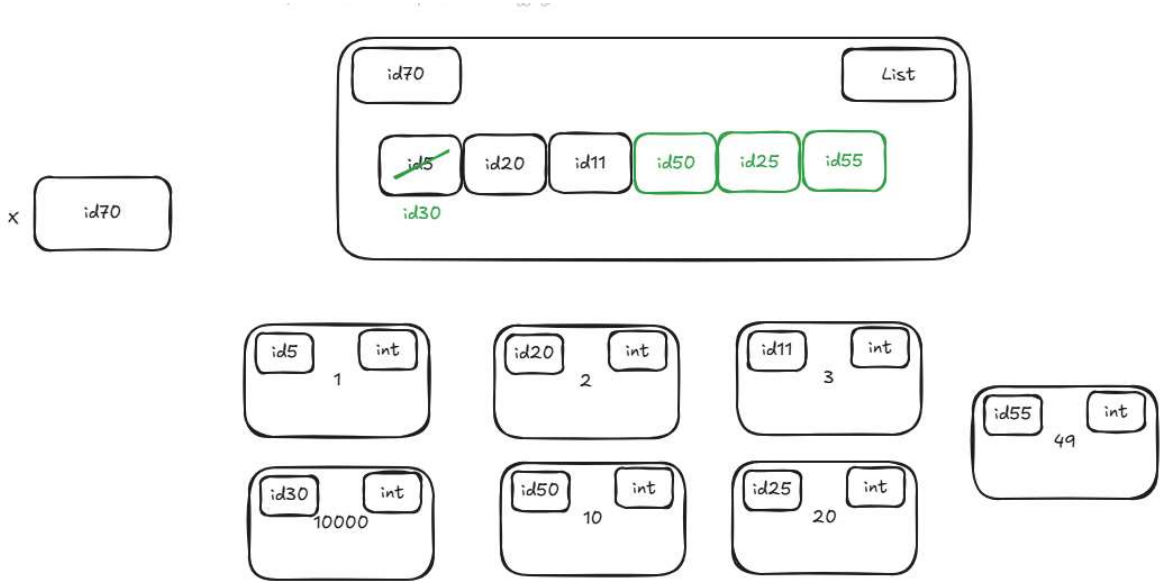
Memory Model (List):

```
>>> x = [1,2,3]
>>> type(x)
<class 'list'>
>>> id(x)
2629899132416
>>>
>>> x[0] = 10000
>>> x
[10000, 2, 3]
>>> id(x)
2629899132416
>>> x.extend([10, 20, 49])
>>> x
[10000, 2, 3, 10, 20, 49]
>>> id(x)
2629899132416
```



- **Tuple**:

- A Tuple is a container used to hold a series of comma-separated values between parenthesis (), such as (x,y) coordinates.
- Tuples are like list, except they are immutable.
- A Tuple is a collection which is ordered and unchangeable.
- The parenthesis is optional, but it is a good practice to use.
- An empty tuple can be created.

```
T = 2,3,4,5
print(T)
(2, 3, 4, 5)
print(type(T))
<class 'tuple'>

T = (2,3,4,5)
print(T)
(2, 3, 4, 5)
print(type(T))
<class 'tuple'>

T = ("SMIT")
print(type(T))
<class 'str'>
T = ("SMIT",)
print(type(T))
<class 'tuple'>
```

- **Tuple Properties:**

A tuple has following properties:

1. Positive and negative indexes.
2. Slicing.

```
T = ("apple", "cherry", "banana")
L = list(T)
L
['apple', 'cherry', 'banana']
print(type(L))
<class 'list'>

Tuple = tuple(L)
Tuple
('apple', 'cherry', 'banana')
print(type(Tuple))
<class 'tuple'>
```

- **Tuple (Unpacking):**

```
fruits = ("apple", "banana", "cherry")
(green, yellow, red) = fruits
print(green)
apple
print(yellow)
banana
print(red)
cherry
```

- **Tuple (Using Asterisk):**

```
fruits = ("apple", "banana", "cherry", "apricot", "mango")
(green, yellow, *red) = fruits
print(green)
apple
print(yellow)
banana
print(red)
['cherry', 'apricot', 'mango']

# Try this statement  (green, *yellow, red) = fruits

(*green, *yellow, red) = fruits
SyntaxError: multiple starred expressions in assignment
```

- **Tuple (Built-in Functions):**

➢ count() → Returns the number of times a specified value occurs in a tuple.

➢ index() → Searches the tuple for a specified value and returns the position of where it was found.

```
fruits
('apple', 'banana', 'cherry', 'apricot', 'mango')
fruits.count("apple")
1
fruits.index("apricot")
3
```

```
# Function example
def my_func(*args):
    print(type(args))
    for arg in args:
        print(arg)
```

```
my_func(1,2,3,4,5)
<class 'tuple'>
1
2
3
4
5
```

- **Dictionary:**

Dictionary in Python is a collection of keys and values, used to store data like a map, unlike other data types which hold only a single value as an element. The dictionary is defined into elements Keys and Values.

- Accessed by keys, not offset position.
- Variable-length, heterogeneous and arbitrarily nestable.
- Mutable mapping.

- **Dictionary (get, keys)**

```
dict
<class 'dict'>
D = {"1": "corolla", "2": "farrari", "3": "alto", "4": "cultus", "5": "1993"}
D
{'1': 'corolla', '2': 'farrari', '3': 'alto', '4': 'cultus', '5': '1993'}
x = D["3"]
print(x)
alto
y = D.get("4")
y
'cultus'
dict_keys = D.keys()
dict_keys
dict_keys(['1', '2', '3', '4', '5'])
# insert element into dictionary
D["6"] = "Alto VXL"
D
{'1': 'corolla', '2': 'farrari', '3': 'alto', '4': 'cultus', '5': '1993', '6'
: 'Alto VXL'}
dict_keys
dict_keys(['1', '2', '3', '4', '5', '6'])
```

```
# Example of get() method in Dictionary
# If key is not present then get() does not throw an error
# and return the None value because default is set to None
words = ['apple', 'cherry', 'apple', 'cherry','raspberry']

freq = {}
for word in words:
    freq[word] = freq.get(word, 0) + 1


print(freq)
{'apple': 2, 'cherry': 2, 'raspberry': 1}
```

- **Dictionary (values, items):**

```
dict_values = D.values()
dict_values
dict_values(['corolla', 'farrari', 'alto', 'cultus', '1993', 'Alto VXL'])
D["year"] = 2022
D
{'1': 'corolla', '2': 'farrari', '3': 'alto', '4': 'cultus', '5': '1993', '6'
: 'Alto VXL', 'year': 2022}
dict_values
dict_values(['corolla', 'farrari', 'alto', 'cultus', '1993', 'Alto VXL', 2022
])

dict_items = D.items()
dict_items
dict_items([('1', 'corolla'), ('2', 'farrari'), ('3', 'alto'), ('4', 'cultus'
), ('5', '1993'), ('6', 'Alto VXL'), ('year', 2022)])
D["year"] = 1947
D
{'1': 'corolla', '2': 'farrari', '3': 'alto', '4': 'cultus', '5': '1993', '6'
: 'Alto VXL', 'year': 1947}
dict_items
dict_items([('1', 'corolla'), ('2', 'farrari'), ('3', 'alto'), ('4', 'cultus'
), ('5', '1993'), ('6', 'Alto VXL'), ('year', 1947)])
```

- **Dictionary (update, pop, popitem)**

```
D
{'1': 'corolla', '2': 'farrari', '3': 'alto', '4': 'cultus', '5': '1993', '6'
: 'Alto VXL', 'year': 1947}
D["4"] = "apple"
D
{'1': 'corolla', '2': 'farrari', '3': 'alto', '4': 'apple', '5': '1993', '6':
'Alto VXL', 'year': 1947}
D.update({"4": "altis"})
D
{'1': 'corolla', '2': 'farrari', '3': 'alto', '4': 'altis', '5': '1993', '6':
'Alto VXL', 'year': 1947}
D.update({"4": "cultus", "3": "honda"})
D
{'1': 'corolla', '2': 'farrari', '3': 'honda', '4': 'cultus', '5': '1993', '6
': 'Alto VXL', 'year': 1947}
# dictionary pop
D.pop("3")
'honda'
D
{'1': 'corolla', '2': 'farrari', '4': 'cultus', '5': '1993', '6': 'Alto VXL',
'year': 1947}
D.popitem()
('year', 1947)
D
{'1': 'corolla', '2': 'farrari', '4': 'cultus', '5': '1993', '6': 'Alto VXL'}
```

- **Dictionary (copy, del, clear):**

```
D_copy = D.copy()
D_copy
{'1': 'corolla', '2': 'farrari', '4': 'cultus', '5': '1993', '6': 'Alto VXL'}
del D['2']
D
{'1': 'corolla', '4': 'cultus', '5': '1993', '6': 'Alto VXL'}
del D
D
Traceback (most recent call last):
  File "<pyshell#621>", line 1, in <module>
    D
NameError: name 'D' is not defined

D_copy.clear()
D_copy
{}
```

- **Dictionary (Built-in functions) [Summary]**

- clear() → Remove all items from the dictionary.
- copy() → Returns a copy of the dictionary.
- get() → Returns the value of the specified key. If key not present then return None (default).
- items() → Returns a list containing a tuple of each key value pair.
- keys() → Returns a list containing dictionary's keys.
- fromkeys() → Returns a dictionary with specified keys and values.
- pop() → Remove the element with specified key.
- popitem() → Removes the last inserted key-value pair.
- update() → Update dictionary with specified key-value pair.
- values() → Returns a list containing dictionary's values.
- setdefault() → Returns the value of the specified key. If the key does not exist: insert the key with specified value.
- del → keyword used to remove the item with the specified key name

Nested Dictionary:

A nested dictionary is a dictionary where the value associated with a key is itself an another dictionary.

```
# creating a nested dictionary for student details
students = {
    'student1': {'name': 'Alice', 'age': 20, 'grade': 'A'},
    'student2': {'name': 'Bob', 'age': 22, 'grade': 'B'},
    'student3': {'name': 'Charlie', 'age': 21, 'grade': 'A+'}
}
print(students)
{'student1': {'name': 'Alice', 'age': 20, 'grade': 'A'}, 'student2': {'name':
'Bob', 'age': 22, 'grade': 'B'}, 'student3': {'name': 'Charlie', 'age': 21, '
grade': 'A+'}}

# access detail of student2
students["student2"]
{'name': 'Bob', 'age': 22, 'grade': 'B'}
# access name of student 2
students["student2"]["name"]
'Bob'
# modify grade of student2
students["student2"]["grade"] = "A-"
print(students)
{'student1': {'name': 'Alice', 'age': 20, 'grade': 'A'}, 'student2': {'name':
'Bob', 'age': 22, 'grade': 'A-'}, 'student3': {'name': 'Charlie', 'age': 21,
'grade': 'A+'}}
# Adding a new student
students['student4'] = {'name': 'David', 'age': 19, 'grade': 'B+'}
print(students['student4'])
SyntaxError: multiple statements found while compiling a single statement
# Adding a new student
students['student4'] = {'name': 'David', 'age': 19, 'grade': 'B+'}
print(students["student4"])
{'name': 'David', 'age': 19, 'grade': 'B+'}
```

Tasks on Dictionary:

1. Create the dictionary:

   {"CP": "Computer Programming", "FCE": "Fundamental of Computer Engineering", "SMIT": "Saylani Mass Information Technology"}

2. Write a statement to add “AI”: “Artificial Intelligence” in the dictionary of Q1.
3. Write program to find out whether the given key (Chinese) is the part of dictionary in Q1.
4. Write a program that takes a list of multiple choice responses for e.g. [‘a’, ‘b’, ‘c’] and prints a dictionary of question response pairs

   {“Q1”: “a”, “Q2”: “b”, “Q3”: “c”}