Programming Tutorial

06 Composite Data

Array, List and Slice

Array

An array[in **Go, Java**] is a **fixed-length** sequence of zero or more elements of a **particulart** type. All the elements are arranged **in line**.

```go
package main

import "fmt"

func main() {
  var a [2]string
  a[0] = "Hello"
  a[1] = "World"
  fmt.Println(a[0], a[1])
  fmt.Println(a)

  primes := [6]int{2, 3, 5, 7, 11, 13}
  fmt.Println(primes)
}
```

List

A list[in **Python**] is a data structure that holds an **ordered** collection of items. Once you have created a list, you can add, remove or search for items in the list. Since we can add and remove items, we say that a list is a **mutable** data type.

```
# This is my shopping  list
shoplist = ['apple', 'mango','carrot', 'banana']

print('I have', len(shoplist), 'items to purchase.')

print('These items are:', end = ' ')
for item in shoplist:
    print(item, end =' ')

print('\nI also have to buy rice.')
shoplist.append('rice')
print('Sorted shopping list is', shoplist)
```

Slice

Slices[in **Go, Python**] represent variable-length sequences whose elements all have the **same** type. A slice is lightweight data structrue that fives access to a **subsequence**(or perhaps all) of the elements of an array.

```
shoplist = ['apple', 'mango','carrot', 'banana']

print('Item 0 is', shoplist[0])
print('Item -1 is', shoplist[-1])
print('Item 1 to 3 is', shoplist[1:3])
print('Item 2 to end is', shoplist[2:])
print('Item 1 to -1 is', shoplist[1:-1])
print('Item start to end is', shoplist[:])
print(shoplist[::1])
print(shoplist[::2])
print(shoplist[::-1])
```

Struct

A struct in the C programming language (and many derivatives) is a **composite** data type (or record) declaration that defines a physically grouped list of variables under one name in a block of memory, allowing the **different** variables to be accessed via a single pointer or by the struct declared name which returns the same address. The struct data type can contain other data types so is used for mixed-data-type records such as a hard-drive directory entry (file length, name, extension, physical address, etc.), or other mixed-type records (name, address, telephone, balance, etc.). The C struct directly references a **contiguous block** of physical memory, usually delimited (sized) by word-length boundaries.

```
/* Forward declare a type "point" to be a struct. */
typedef struct point point;
```

```
/* Declare the struct with integer members x, y */
struct point {
   int    x;
   int    y;
};
int main(void)
{
    point p = { 1, 3 };        /* initialized variable */
    point q;                   /* uninitialized */
    q = p;                     /* copy member values from p into q */
    return 0;
}
```

Class and Object

A **class** creates a new *type* where **objects** are **instances** of the class. An analogy is that you can have variables of type int which translates to saying that variables that store integers are variables which are instances (objects) of the int class.

Objects can store data in **different** types using ordinary variables that *belong* to the object. Variables that belong to an object or class are referred to as **fields**. Objects can also have functionality by using functions that *belong* to a class. Such functions are called **methods** of the class. This terminology is important because it helps us to differentiate between functions and variables which are independent and those which belong to a class or object.

```python
class Point:
    def __init__(self, x=0, y=0):
        self.x, self.y = x, y
    def set(self, x, y):
        self.x, self.y = x, y
p = Point(10, 10)
p.set(0, 0)
print(p.x, p.y)
Point.set(p, 0, 1)
print(p.x, p.y)
```

Tuple, Dictionary

Tuple

Tuples are used to hold together multiple objects. Think of them as similar to lists, but without the extensive functionality that the list class gives you. One major feature of tuples is that they are **immutable** like strings i.e. you cannot modify tuples.

```
zoo = ('python', 'elephant','penguin')
print('Number of animals in the zoo is', len(zoo))

new_zoo = 'monkey', 'camel', zoo
print('Number of animals in the zoo is', len(new_zoo))
print('All animals in new zoo are', new_zoo)
print('Animals brought from old zoo are', new_zoo[2])
```

Dictionary

A dictionary is like an address-book where you can find the address or contact details of a person by knowing only his/her name i.e. we associate **keys** (name) with **values** (details). Note that the key must be unique just like you cannot find out the correct information if you have two persons with the exact same name.

Note that you can use only immutable objects (like strings) for the keys of a dictionary but you can use either immutable or mutable objects for the values of the dictionary. This basically translates to say that you should use only simple objects for keys.

```
ab = {
    'Swaroop':'swaroop@swaroopch.com',
    'Larry':'larry@wall.org',
    'Matsumoto':'matz@ruby-lang.org',
    'Spammer':'spammer@hotmail.com'
}

print("Swarrop's address is", ab['Swaroop'])

for name, address in ab.items():
    print('Contact {} at {}'.format(name, address))
```