AN7914 Week 03 Python

February 11, 2024

1 Week 3 Python

We have looked at storing information in a variable. Like the examples below.

[1]:
```
a='Sakib'
b=1
c=2.0
```

Maybe we want to store the age of Sakib. We could do the following.

[2]:
```
sakib_age=100
```

But this is not ideal if we have to store age of everyone in this class. We would need atleast 16 variables. Waste of memory and engergy! We can instead store more information using python's builtin **Data Structures**. See one example below. We have stored age of 5 individuals in one go.

[3]:
```
age=[10,20,32,35,26]
```

1.1 Data Structures

Python have the following builtin **Data Structure**. - list - dictionary - tuple - sets

- lists are usually written by wrapping items with `[]`.
- dictionary are usually written by wrapping items with `{}`.
- tuple are usually written by wrapping items with `()`.

Tuple

[4]:
```
age_tuple=(10,20,30,200)
```

Dictionary

[5]:
```
students={'Hermoine': 'Gryffindor',
          'Harry' : 'Gryffindor',
          'Ron': 'Gryffindor',
          'Draco' : 'Slytherin'}
```

Lists

[6]:
```
age=[10,20,32,35,26]
```

We will rarely use sets. So we skip this for now.

1.2 Lists

Let's create a list of students.

[7]:
```
list_of_student=['Hermoine','Harry','Ron','Draco']
```

Now let's create a list of animals

[8]:
```
animals=['dog', 'cat', 'cow']
```

We previously created a list of age:

`age=[10,20,32,35,26]`

The following list contains a string, a float, an integer, and another list!:

[9]:
```
anything=['spam', 2.0, 5, [10, 20]]
```

A list within another list is nested. A list that contains no items is called an empty list; you can create one with empty brackets, `[]`

[10]:
```
empty = []
```

You can find out how many items does a list have. In other words what is the length of a string. We can use the function `len()`. Let's find out how many items does the `list_of_student` of have.

[11]:
```
len(list_of_student)
```

[11]:
```
4
```

We can access specific items of list. To find the first item we do the following.

[12]:
```
list_of_student[0]
```

[12]:
```
'Hermoine'
```

[13]:
```
print(list_of_student[0])
```

```
Hermoine
```

Notice we use `0`. In python and in most programming language indexing starts from 0. To find the last item or the index number 3 we do the following.

[14]:
```
list_of_student[3]
```

[14]:
```
'Draco'
```

What if you wanted to access index 4 or the 5th item in the list? You could try the following! But this won't work, because our list only contains 4 items. So you get an error!

[15]:
```
list_of_student[4]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[15], line 1
----> 1 list_of_student[4]

IndexError: list index out of range
```

We can also append another item to the list by using the `append()` functin/method.

[16]:
```
list_of_student.append('Sakib')
```

[17]:
```
list_of_student
```

[17]:
```
['Hermoine', 'Harry', 'Ron', 'Draco', 'Sakib']
```

Lists are **mutable**, because you can change the order of items in a list or reassign an item in a list. Let us replace `Sakib` with `Joe` in the list.

[18]:
```
list_of_student[4]='Joe'
list_of_student
```

[18]:
```
['Hermoine', 'Harry', 'Ron', 'Draco', 'Joe']
```

If we want print all the students name one by one we could do the following.

[19]:
```
print(list_of_student[0])
print(list_of_student[1])
print(list_of_student[2])
print(list_of_student[3])
print(list_of_student[4])
```

```
Hermoine
Harry
Ron
Draco
Joe
```

But we can improve this further by using a loop to iterate over the list.

[20]:
```
for x in list_of_student:
    print(x)
```

```
Hermoine
Harry
Ron
```

```
Draco
Joe
```

Imagine that you don't simply want to print the name of the student, but also their position in the list. To accomplish this, you can edit your code as follows:

[21]:
```
for i in range(len(list_of_student)):
    print(i+1,list_of_student[i])
```

```
1 Hermoine
2 Harry
3 Ron
4 Draco
5 Joe
```

1.3 Dictionary

`dicts` or dictionaries is a data structure that allows you to associate *keys* with *values*. Where a list is a list of multiple values, a dict associates a key with a value. Considering the houses of Hogwarts, we might assign specific students to specific houses.

Hermoine	Harry	Ron	Draco
Gryffindor	Gryffindor	Gryffindor	Slytherin

We could use lists alone to accomplish this:

[22]:
```
students = ["Hermoine", "Harry", "Ron", "Draco"]
houses = ["Gryffindor", "Gryffindor", "Griffindor", "Slytherin"]
```

Notice that we could promise that we will always keep these lists in order. The individual at the first position of students is associated with the house at the first position of the houses list, and so on. However, this can become quite cumbersome as our lists grow!

[23]:
```
students = {
    "Hermoine": "Gryffindor",
    "Harry": "Gryffindor",
    "Ron": "Gryffindor",
    "Draco": "Slytherin",
}
print(students["Hermoine"])
print(students["Harry"])
print(students["Ron"])
print(students["Draco"])
```

```
Gryffindor
Gryffindor
Gryffindor
Slytherin
```

Notice how we use `{}` curly braces to create a dictionary. Where `lists` use numbers to iterate through the list, `dicts` allow us to use words.

A *dictionary* is like a list, but more general. In a list, the index positions have to be integers; in a dictionary, the indices can be (almost) any type. You can think of a dictionary as a mapping between a set of indices (which are called *keys*) and a set of *values*. Each key maps to a value. The association of a key and a value is called a **key-value** pair or sometimes an item.

In the example above `"Hermoine"` is a key and `"Gryffindor"` is a value. To access a specific value we do the following.

[24]:
```
print(students['Hermoine'])
```

```
Gryffindor
```

[25]:
```
print(students['Draco'])
```

```
Slytherin
```

If we we had to print all the values one by one like we did couple of cells ago it would be tedious and time consuming. We can improve by doing this.

[26]:
```
for name in students:
    print(name)
```

```
Hermoine
Harry
Ron
Draco
```

Notice how executing this code, the for loop will only iterate through all the keys, resulting in a list of the names of the students. How could we print out both values and keys?

[27]:
```
for name in students:
    print(students[name])
```

```
Gryffindor
Gryffindor
Gryffindor
Slytherin
```

Modify your code as follows:

[28]:
```
for name in students:
    print(name, students[name])
```

```
Hermoine Gryffindor
Harry Gryffindor
Ron Gryffindor
Draco Slytherin
```

If we want to print something more useful we can do the following.

[29]:
```
for name in students:
    print(name,"lives in", students[name])
```

```
Hermoine lives in Gryffindor
Harry lives in Gryffindor
Ron lives in Gryffindor
Draco lives in Slytherin
```

What if we have more information about our students? How could we associate more data with each of the students?

Name	House	Gender
Hermoine	Gryffindor	Female
Harry	Gryffindor	Male
Draco	Slytherin	Male

You can imagine wanting to have lots of data associated multiple things with one key. Enhance your code as follows:

[30]:
```
students_info=[
    {'name': 'Hermoine', 'house':'Gryffindor', 'gender':'female'},
    {'name' : 'Hary', 'house':'Gryffindor', 'gender' :'male'},
    {'name': 'Draco', 'house': 'Slytherin', 'gender': 'male'}
]
```

Notice how this code creates a `list` of `dicts`. The `list` called `students_info` has three `dicts` within it: One for each student. Now, you have access to a whole host of interesting data about these students.

[31]:
```
for student in students_info:
    print(student['name'],'lives in',
          student['house'], 'and he/she is a',
          student['gender']
         )
```

```
Hermoine lives in Gryffindor and he/she is a female
Hary lives in Gryffindor and he/she is a male
Draco lives in Slytherin and he/she is a male
```

If you want to add a new item in a dictionary you can just assign a new value to a key. Let's take a loot at our students list first. Then we will add Sakib to the list.

[32]:
```
students
```

[32]:
```
{'Hermoine': 'Gryffindor',
 'Harry': 'Gryffindor',
 'Ron': 'Gryffindor',
 'Draco': 'Slytherin'}
```

[33]:
```
students['sakib']='Winchester'
```

[34]:
```
students
```

[34]:
```
{'Hermoine': 'Gryffindor',
 'Harry': 'Gryffindor',
 'Ron': 'Gryffindor',
 'Draco': 'Slytherin',
 'sakib': 'Winchester'}
```

[35]:
```
students['Draco']='Winchester'
```

[36]:
```
students
```

[36]:
```
{'Hermoine': 'Gryffindor',
 'Harry': 'Gryffindor',
 'Ron': 'Gryffindor',
 'Draco': 'Winchester',
 'sakib': 'Winchester'}
```

[ ]:
```

```