ITP 115 – Programming in Python

Dictionaries

Recall: Lists

- Ordered sequence of things
- Access items by index

```
words = ["doctor", "juan", 47]
```

0	1	2
doctor	juan	47

```
print(words[1])
"juan"
```

Dictionaries

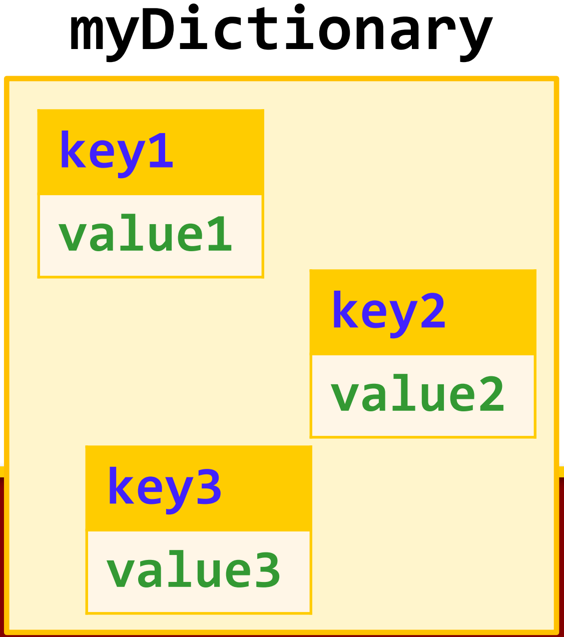
- Store information in pairs
  - a **key** and its **value**
- Like an actual dictionary where each entry is a pair
  - a **word** and its **definition**

Dictionary Syntax

- Define dictionary with `{ }`

- Include `key:value` pairs separated by `,`

```
myDictionary = {key1:value1, key2:value2,
      key3:value3}
```



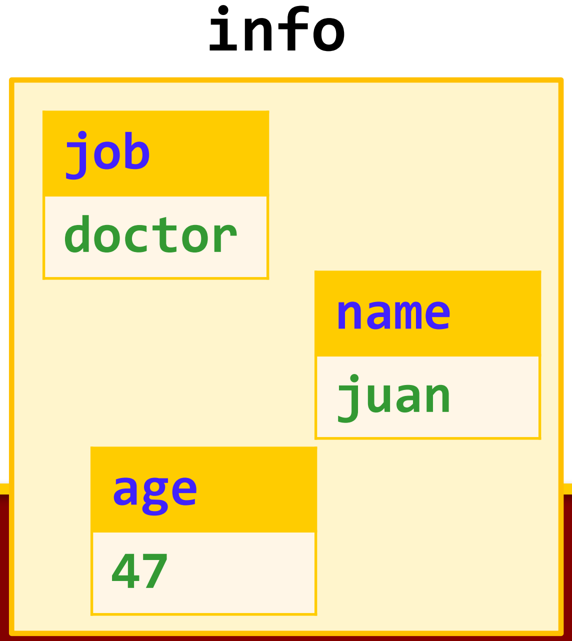
Dictionaries vs. Lists

- Access items by key (*dictionary*), instead of position (*list*)

```
info = {"name":"juan","age":47,"job":"doctor"}
print(info["name"])
```

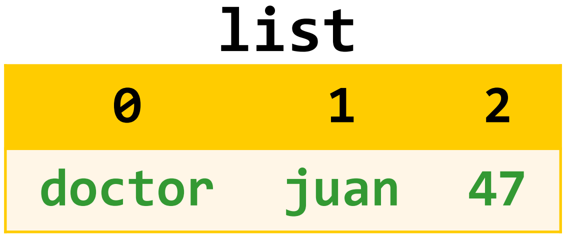
```
                    "juan"
```

- Not ordered sequentially



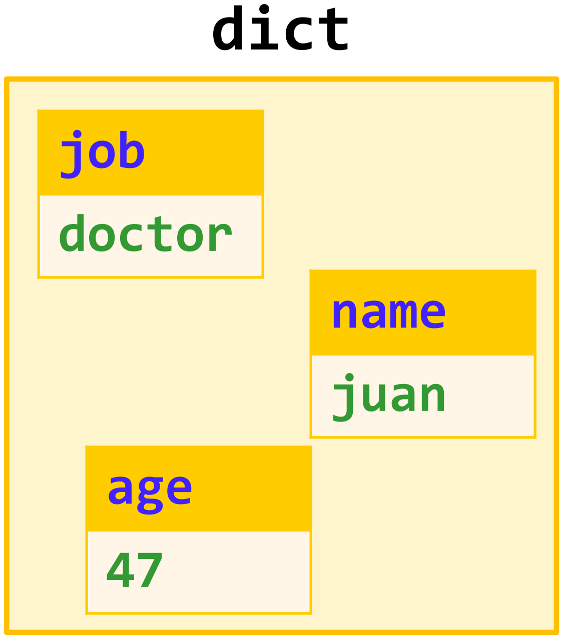
Advantages of Dictionaries

- Dictionaries store vast amounts of data
- Retrieval of information must be *efficient*
- ***Logical*** structure of data is more important than ***physical*** structure of data (in memory)



list

0	1	2
doctor	juan	47



dict

job	name	age
doctor	juan	47

Dictionary Requirements

- Keys **must be** unique
  - Old value will be replaced with new value

- Keys **must be** immutable
  - string, number or tuple

- Values are **not** have to be unique

- Value can be immutable or mutable

Dictionary Keys

- Has UNIQUE keys (only one of each)
- But…can have different keys with the same value

Valid

•Unique keys

•Repeated values

```
info = {"name"    :"juan",
        "nickname":"juan",
           ...
       }
```

Invalid

•Repeated keys

```
info = {"name"    :"juan",
        "name"    :"john",
       }
```

Dictionaries

- Create a dictionary
- Access values
  - Use a key to retrieve a value
  - Testing for a key with the `in` operator before retrieving a value
  - Use the `get()` method to retrieve a value
- Add a key-value pair
- Replace a key-value pair
- Delete a key-value pair

Dictionary Methods

Method	Description
`len(dict)`	Returns number of entries in dict
`dict.get(key, [default])`	Returns the value of key. If key doesn't exist, then the optional default is returned. If key doesn't exist and default isn't specified, then None is returned.
`dict.pop(key, [default])`	Removes the key and returns the value. If key doesn't exist, then the optional default is returned. If key doesn't exist and default isn't specified, then None is returned.
`dict.keys()`	Returns a list of all the keys in a dictionary.
`dict.values()`	Returns a list of all the values in a dictionary.
`dict.items()`	Returns a list of all the items in a dictionary. Each items is a two-element tuple (key, value)
`del dict[key]`	Removes the key. If key doesn't exist, then an error is generated.

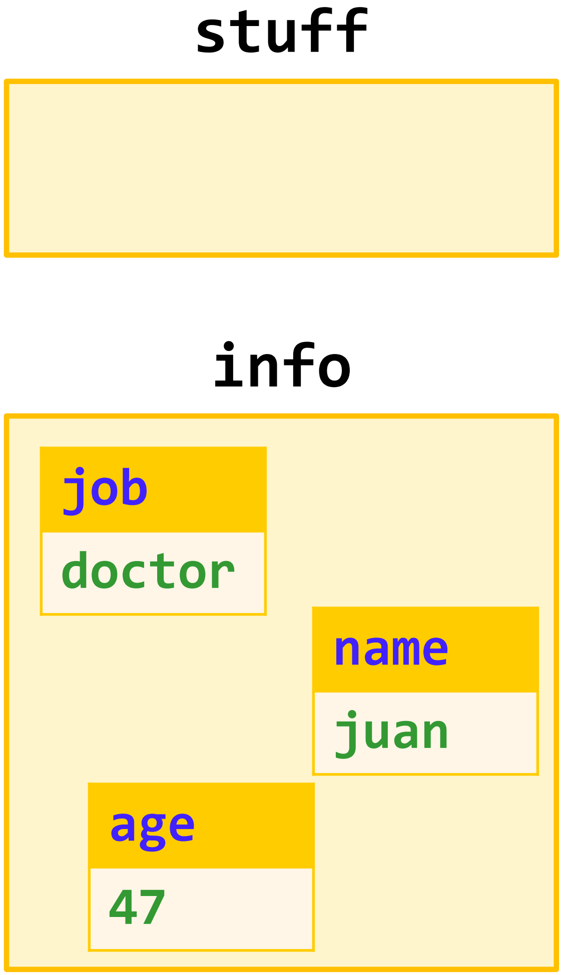
Creating Dictionaries

- Create empty dictionary

```
stuff = { }
```

- Create dictionary with keys

```
info = {
    "name":"juan",
    "age":47,
    "job":"doctor"
}
```



stuff

info

job
doctor

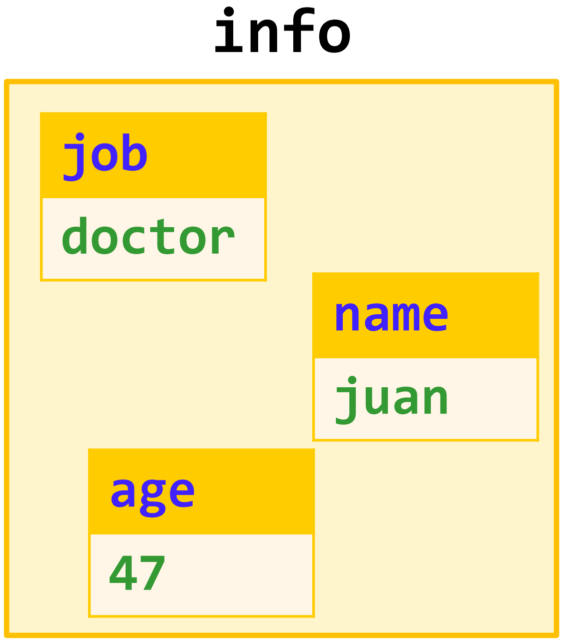
name
juan

age
47

Reading from Dictionary

- Use key

```
username = info["name"]
#username has value "juan"
```



info

job
doctor

name
juan

age
47

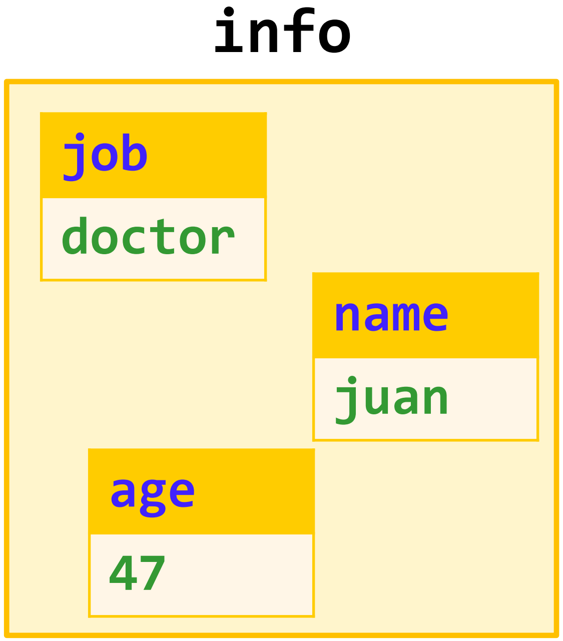
Reading from Dictionary

- Error if key is not in dictionary

```
username = info["hobby"]
#error - key doesn't exist
```

- Error if you use value

```
username = info["juan"]
#error - key doesn't exist
```



info

job
doctor

name
juan

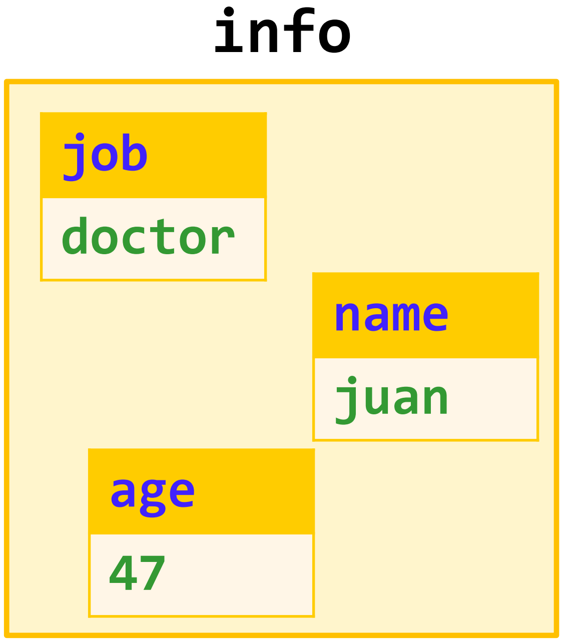
age
47

Reading from Dictionary

- With **for** loop

```
for key in info:
   print(info[key])
```

```
doctor
47
juan
```



info

job

doctor

name

juan

age

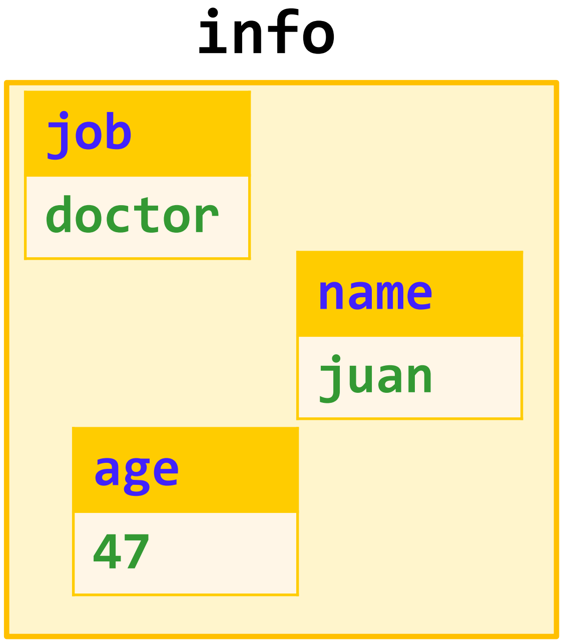
47

Reading from Dictionary

- With **for** loop

```
for key in info:
  print(key, ":", info[key])
```

```
job : doctor
name : juan
age : 47
```



info

job

doctor

name

juan

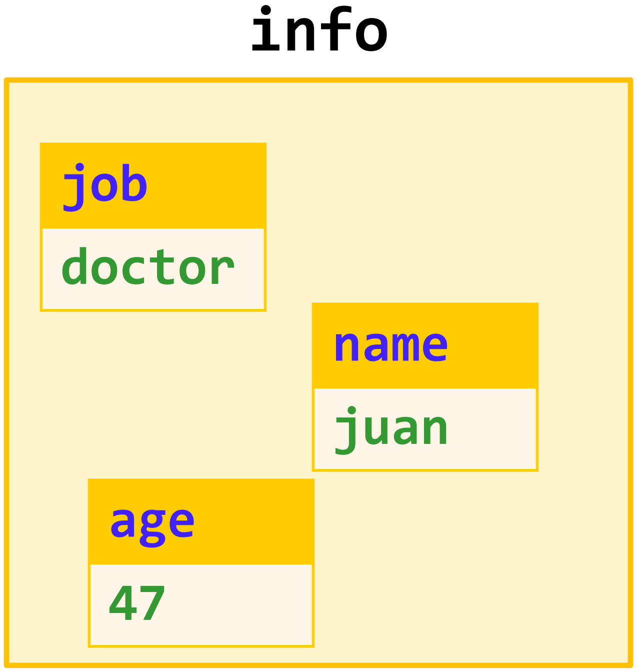
age

47

Adding Key/Values to Dictionary

- Use key

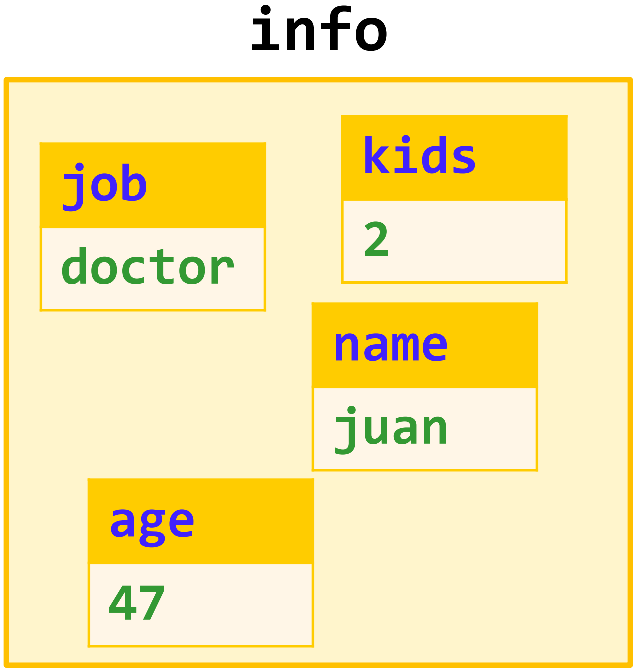
```
info["kids"] = 2
```



Adding Key/Values to Dictionary

- Use key

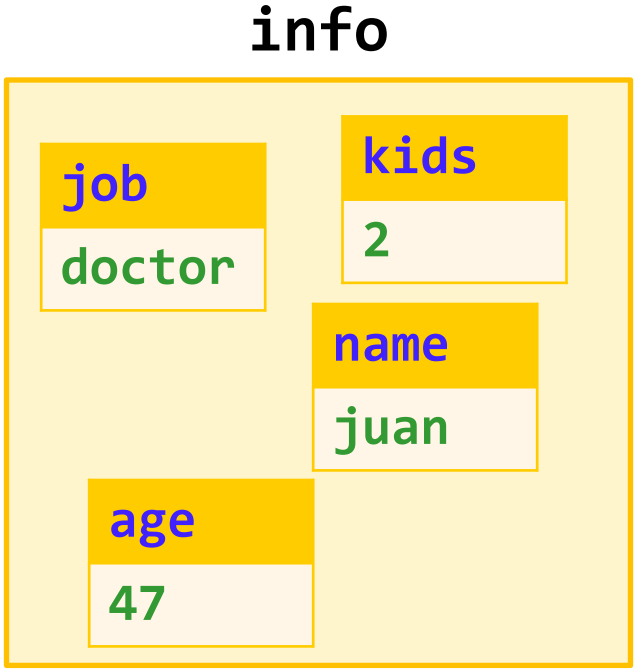
```
info["kids"] = 2
```



Replacing Key/Values

- Use key

```
info["age"] = 34
```

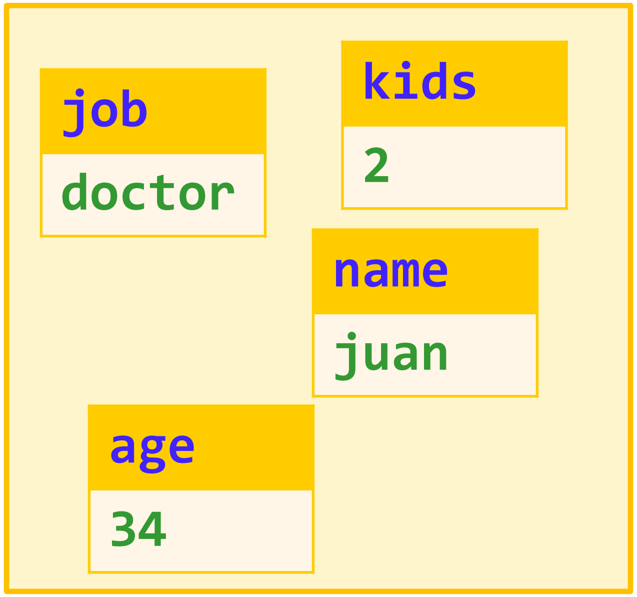


Replacing Key/Values

- Use key

```
info["age"] = 34
```

info



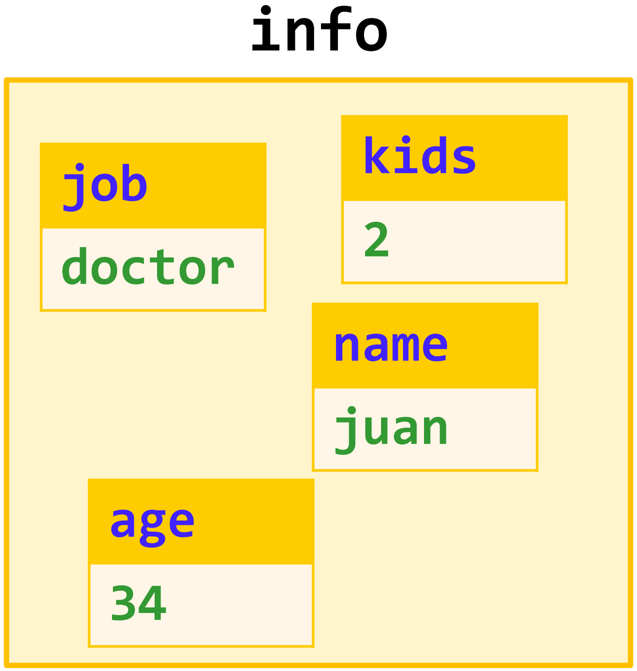
Common Dictionary Operations

- Size of dictionary

```
size = len(info)
#size is 4
```

- Check for keys

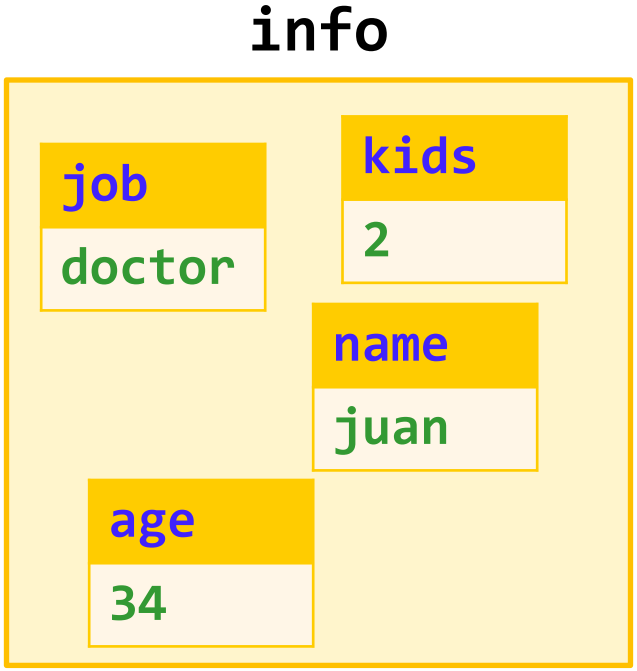
```
if "age" in info:
   print("Found key age")
```



Deleting Keys

- Use key

```
del info["kids"]
```

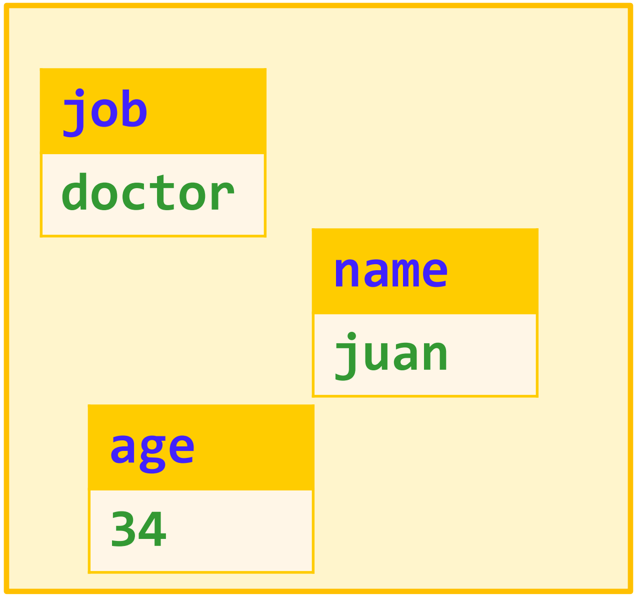


Deleting Keys

- Use key

```
del info["kids"]
```

info



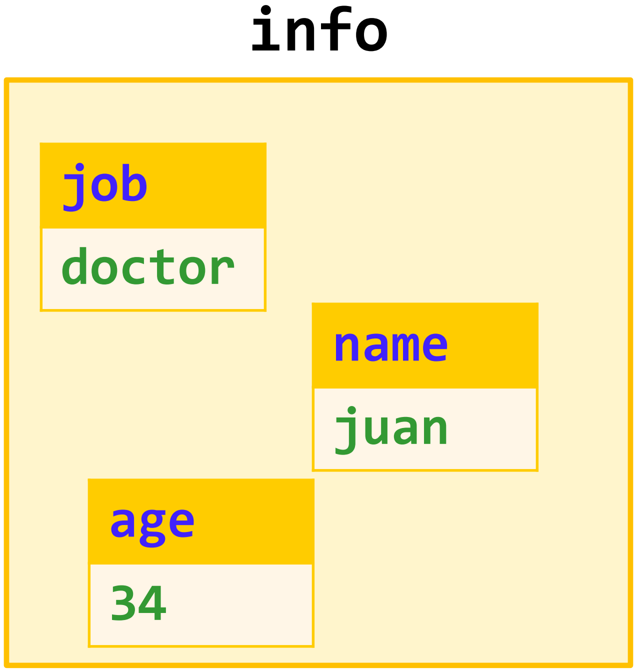
Deleting Keys

- Always check if key exist first

```
del info["hobby"]
#Error! Key not found
```

- Instead…

```
if "hobby" in info:
    del info["hobby"]
#Not found, but no error
```



info

job
doctor

name
juan

age
34

Related Operations

- Get **list** of *all* keys

```
keyList = info.keys()
print(keyList)
```

Output

```
["job", "age", "name"]
```

- Get **list** of *all* values

```
valuesList = info.values()
print(valuesList)
```

Output

```
["juan", 47, "doctor"]
```

Time Comparison

- Data structures
  - List
  - Tuple
  - Dictionary
  - Set (think *"a collection with no duplicate items"*)
- Compare time to store 11 million words from a text file into a data structure
- Compare time to check if a randomly selected word is in data structure

Time Comparison

- Demo

Reference only

Sets

- Mutable data type
- Store unordered, non-duplicate values
  - Like **sets** in math

Reference only

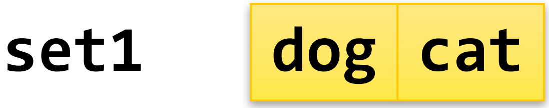
Set Operation

- Create empty set

```
set1 = set()
```

- Create set

```
set1 = {"dog", "cat"}
```

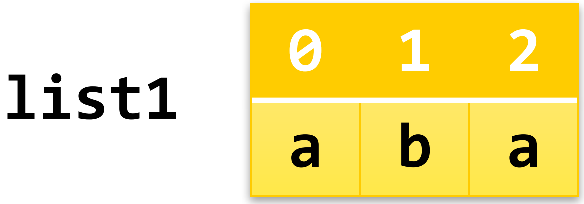


set1

dog	cat

- Create set from list

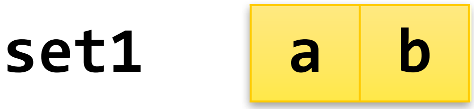
```
list1 = ["a", "b", "a"]
```



list1

0	1	2
a	b	a

```
set1 = set(list1)
```



set1

a	b

Reference only

Set Operation

```
set1 = {"a", "b"}
```

- Length

```
l = len(s)
```

- Check for element

```
if "a" in s:
    print("Found it!")
```

Reference only

Set Operation

```
set1 = {"a", "b"}
```

set1	a	b

- Add elements

```
set1.add("a")
set1.add("x")
```

set1	a	b	
set1	a	b	x

- Remove elements

```
if "a" in set1:
    set1.remove("a")
```

set1	b	x

Always check if element is in set

Reference only

Set Methods

`setA` 1 2 3 `setB` 3 4

Method	Syntax	Result	Description
union	`setA | setB`	`{1, 2, 3, 4}`	Set of elements in **either** `setA` **or** `setB`
intersection	`setA & setB`	`{3}`	Set of elements in **both** `setA` **and** `setB`
difference	`setA - setB`	`{1, 2}`	Set of elements in `setA`, **but not** `setB`
symmetric difference	`setA ^ setB`	`{1, 2, 4}`	Set of elements in `setA` **or** `setB`, **but not both**

Reference only

Sets vs. Lists

- Lists can have duplicate elements; sets do not
- Lists are ordered; sets are unordered
- Checking if an element is in set is *significantly faster* than in a list
  - Sets are **hashable**
- Hash functions are very efficient methods of retrieving data