Python 101

Dictionaries

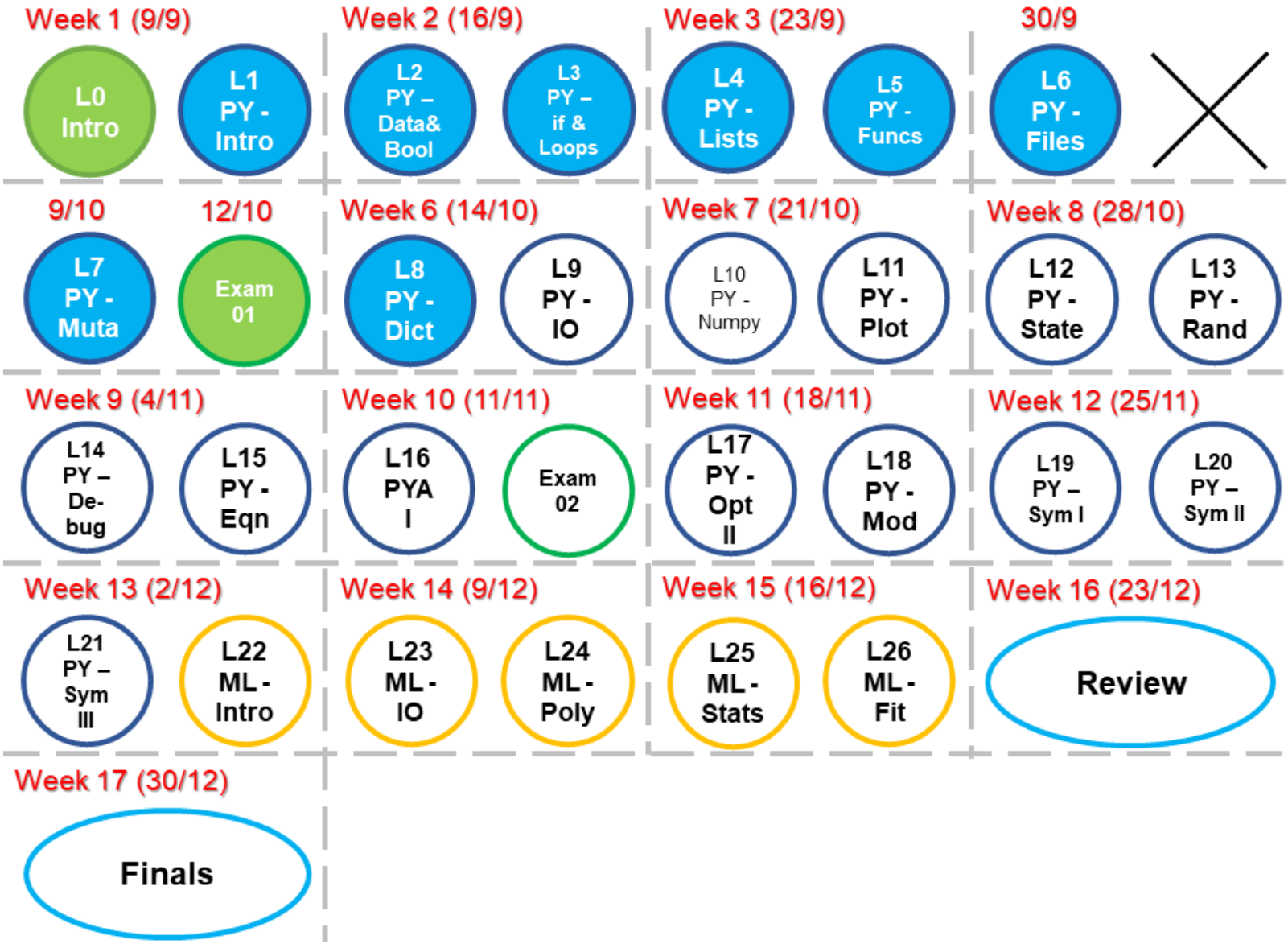
Announcements

quiz: `quiz08` due on Tues 15/10

lab: `lab05` will be on Fri 18/10

hw: `hw04` due Wed 16/10

Roadmap

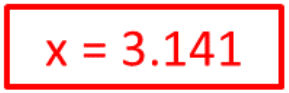

Week 1 (9/9)
L0 Intro
L1 PY - Intro
Week 2 (16/9)
L2 PY – Data& Bool
L3 PY – if & Loops
Week 3 (23/9)
L4 PY - Lists
L5 PY - Funcs
30/9
L6 PY - Files
9/10
L7 PY - Muta
12/10
Exam 01
Week 6 (14/10)
L8 PY - Dict
L9 PY - IO
Week 7 (21/10)
L10 PY - Numpy
L11 PY - Plot
Week 8 (28/10)
L12 PY - State
L13 PY - Rand
Week 9 (4/11)
L14 PY – De-bug
L15 PY - Eqn
Week 10 (11/11)
L16 PYA I
Exam 02
Week 11 (18/11)
L17 PY - Opt II
L18 PY - Mod
Week 12 (25/11)
L19 PY – Sym I
L20 PY – Sym II
Week 13 (2/12)
L21 PY – Sym III
L22 ML - Intro
Week 14 (9/12)
L23 ML - IO
L24 ML - Poly
Week 15 (16/12)
L25 ML - Stats
L26 ML - Fit
Week 16 (23/12)
Review
Week 17 (30/12)
Finals

Objectives

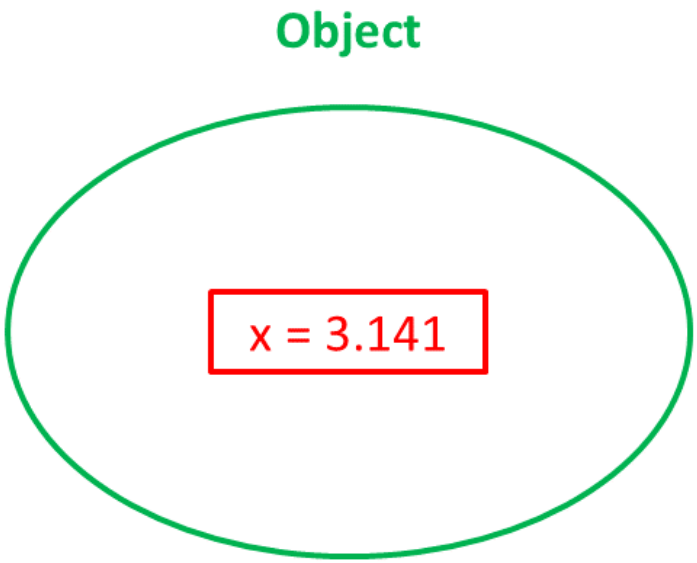
A. Access items in a nested `list`. => expand *lec04 List*
B. Explain how the `dict` associates keys with values.
C. Use `dicts` as accumulators with loops. => expand *lec03 Loops*

Mutability Recap

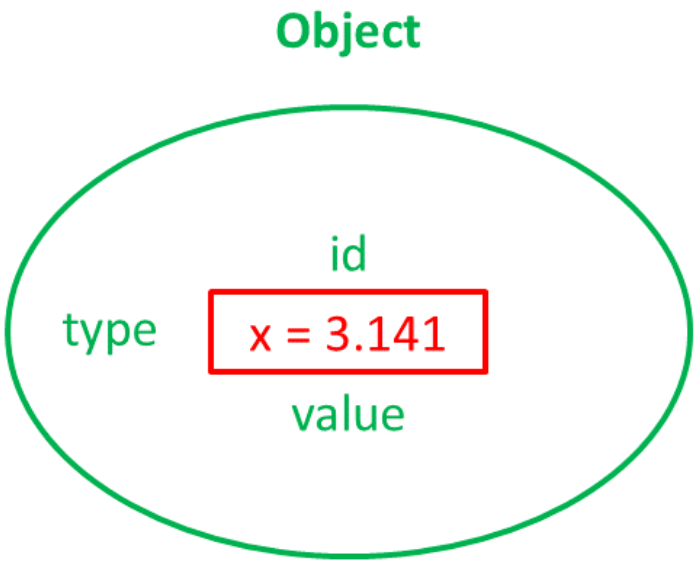
Object



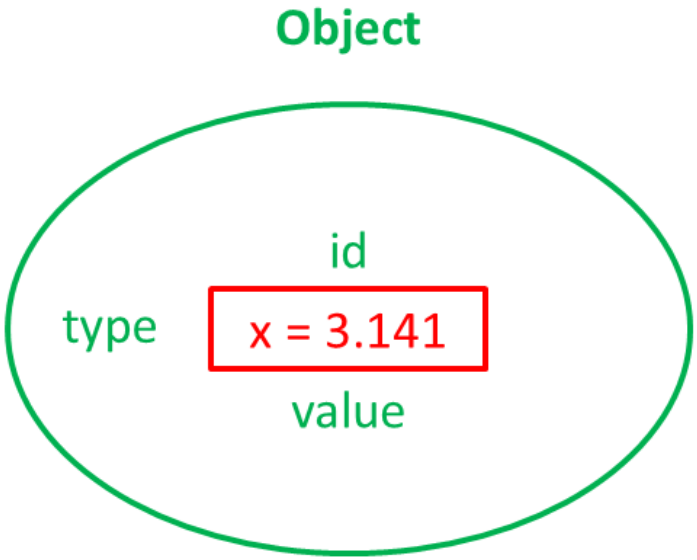
Object



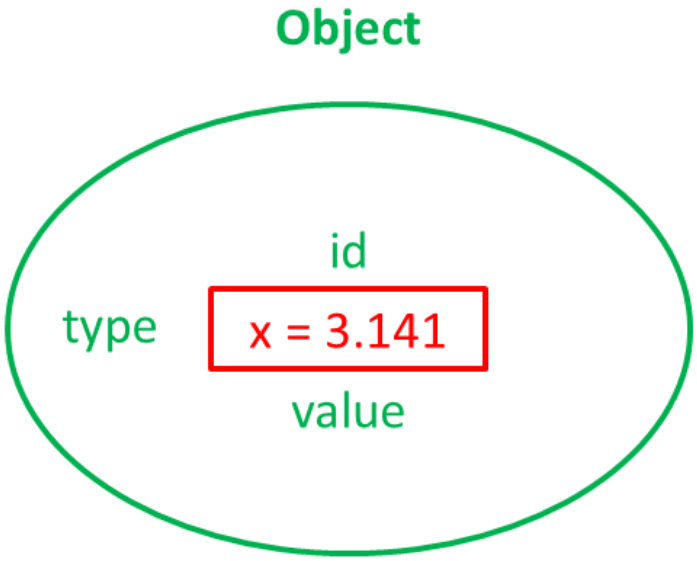
Object

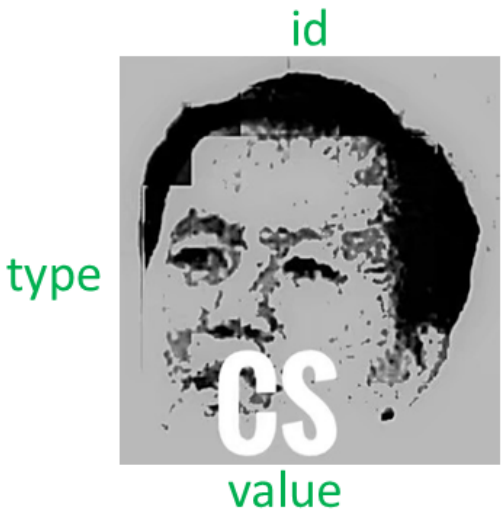


Object

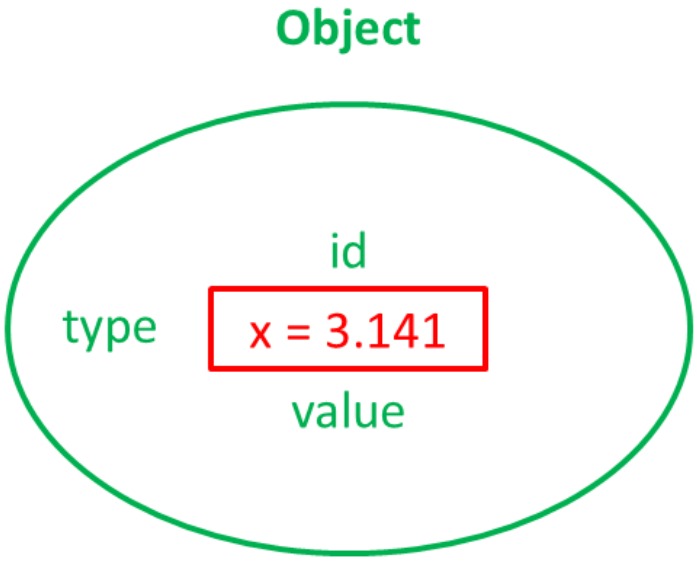


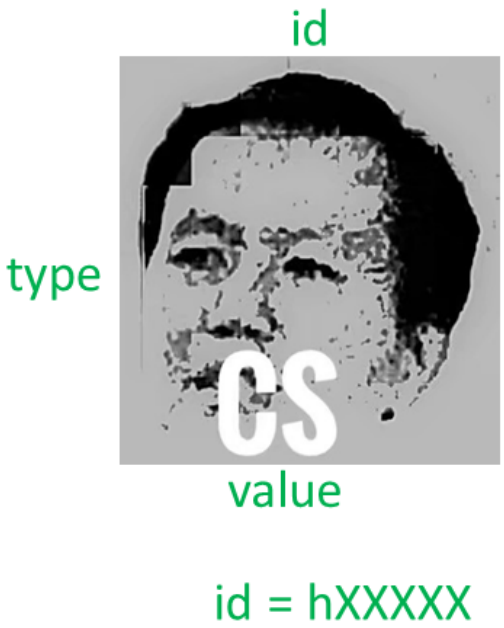
Object



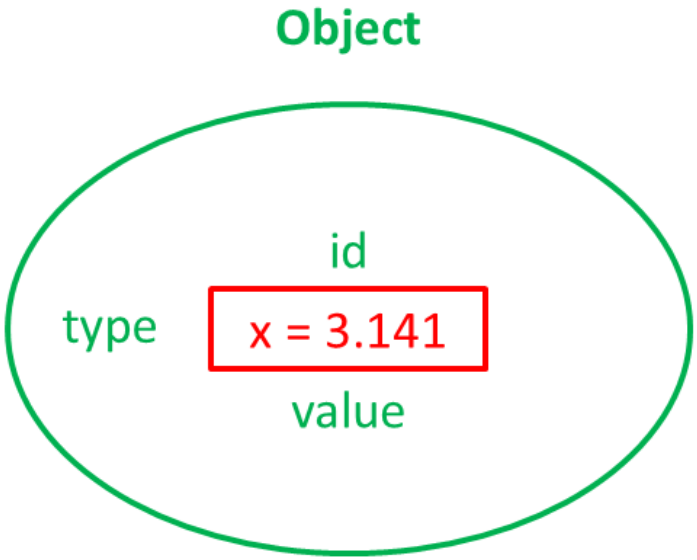


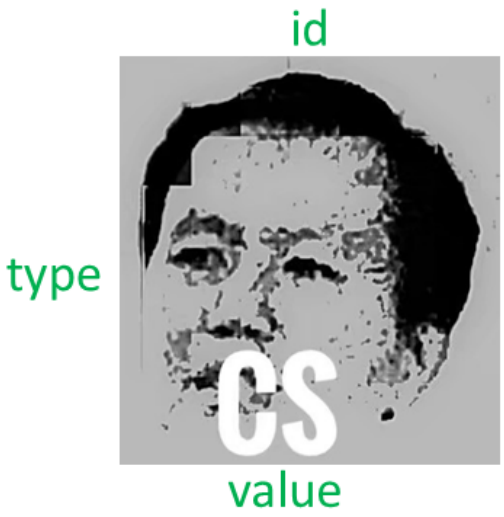
Object





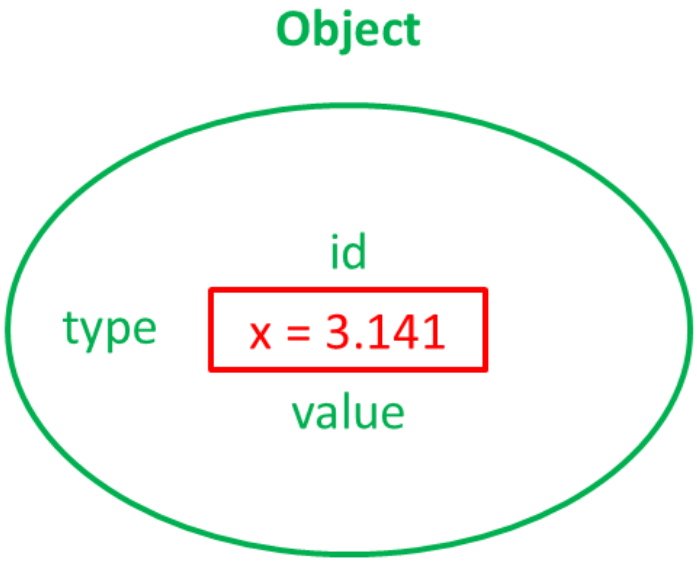
Object

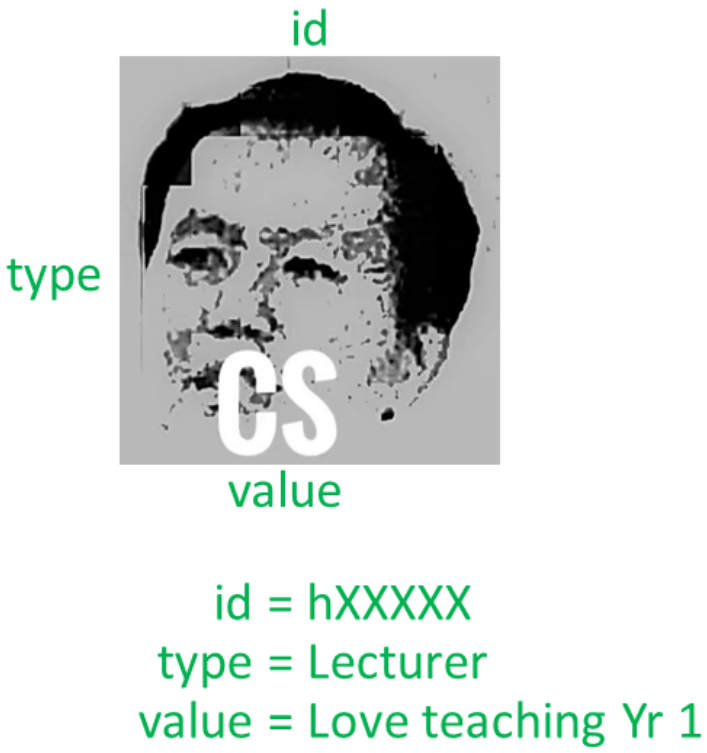




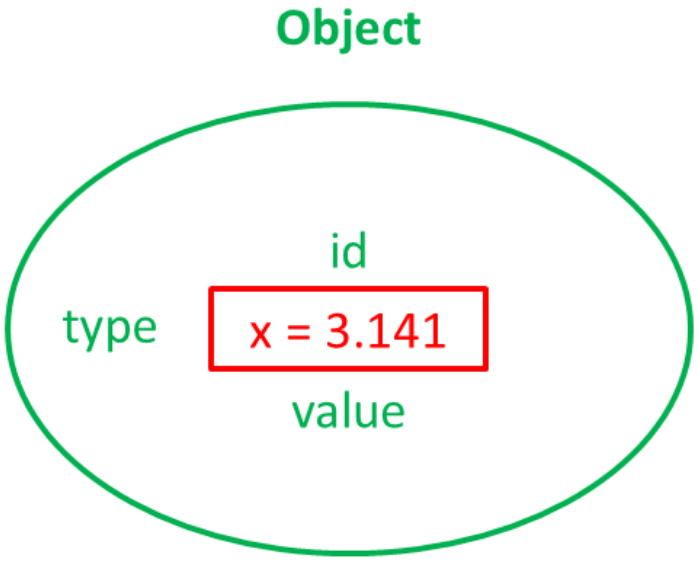
id = hXXXXX
type = Lecturer

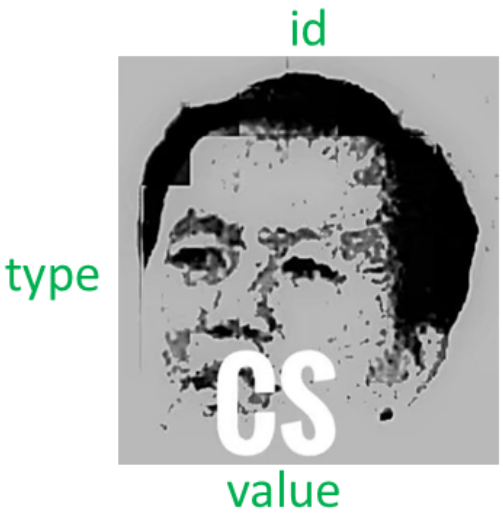
Object


Object
id
type
x = 3.141
value


id
type
value
id = hXXXXX
type = Lecturer
value = Love teaching Yr 1

Object

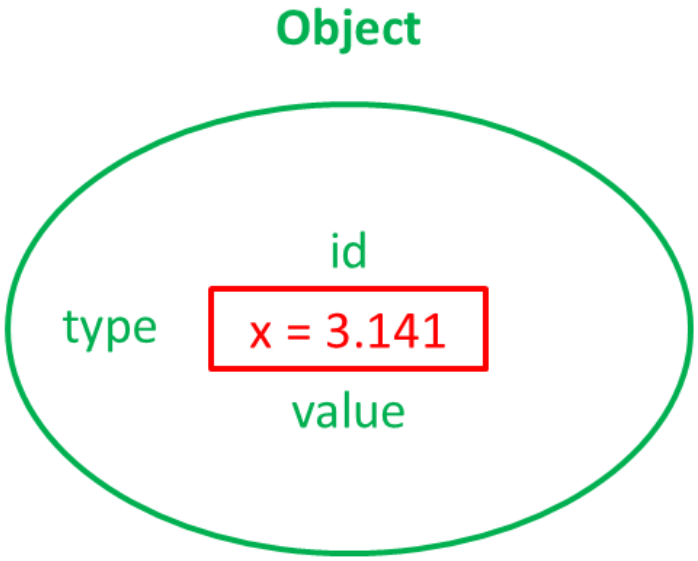


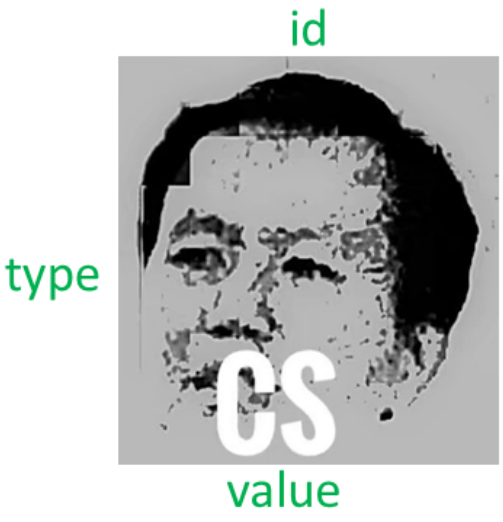


id of x = address of x - some numbers

id = hXXXXX
type = Lecturer
value = Love teaching Yr 1

Object

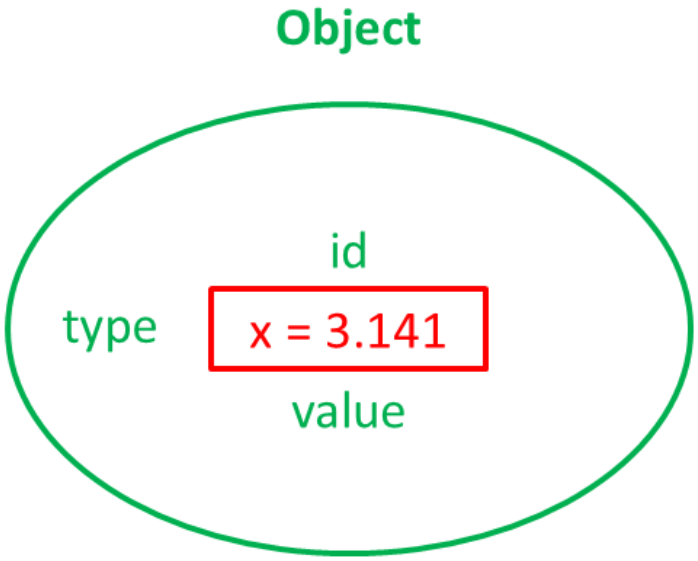


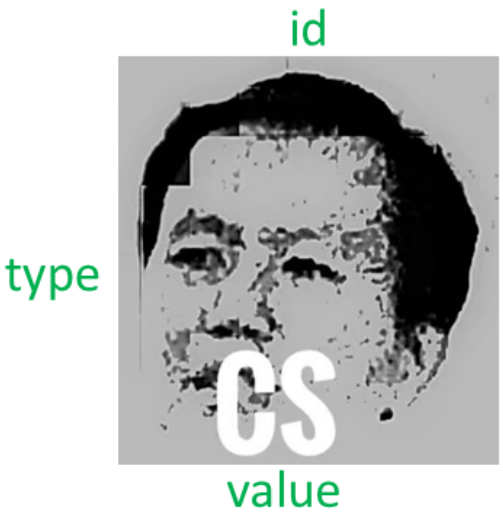


id of x = address of x - some numbers
type of x = float

id = hXXXXX
type = Lecturer
value = Love teaching Yr 1

Object





id of x = address of x - some numbers
type of x = float
value of x = 3.141

id = hXXXXX
type = Lecturer
value = Love teaching Yr 1

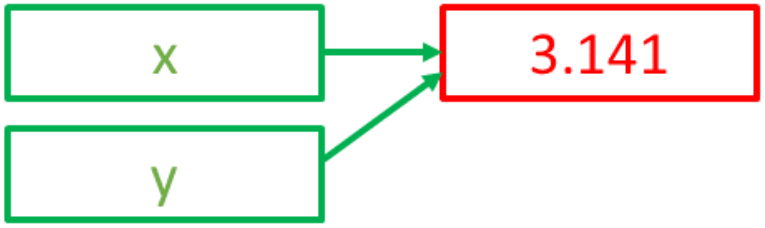
Immutability

Immutability describes objects with values that are not changeable.

Basic python data types - `str`, `int`, `float`, `complex` are *immutable*

Other immutable types - `tuple`...

```
x = 3.141

y = x
```



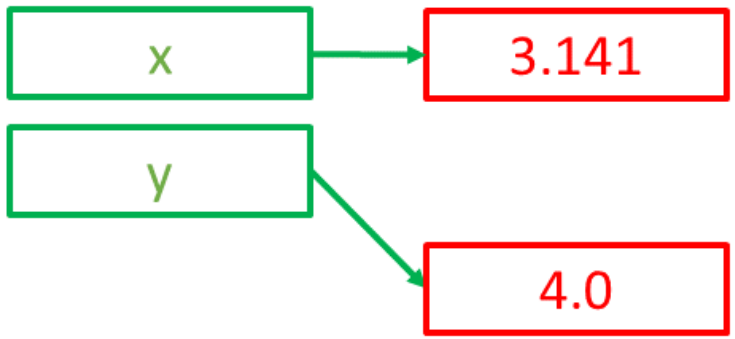
Immutability

Immutability describes objects with values that are not changeable.

Basic python data types - `str`, `int`, `float`, `complex` are *immutable*

Other immutable types - `tuple`...

```
x = 3.141

y = x

y = 4.0
```

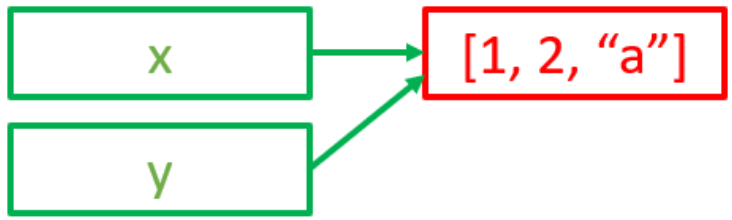


Mutability

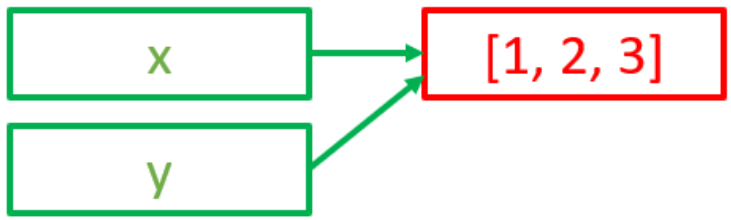
Mutability describes objects with values that are changeable.

python types - `list`, `dict`, are *mutable*

```
x = [1, 2, "a"]

y = x
```




```
x = [1, 2, "a"]

y = x

y[2] = 3
```



id( )

`id(k)` command gives you that unique id number corresponding to object referred to by *k*.

```
x = [ 'a','b','c','d' ]
y = x
z = x[ : ]
id(x)
id(y)
id(z)
```

id( )

`id(k)` command gives you that unique id number corresponding to object referred to by *k*.

```
x = [ 'a','b','c','d' ]
y = x
z = x[ : ]
id(x)
id(y)
id(z)
```

```
id(x) = id(y) = 1743200238
id(z) = 1743204085 # different
```

Note: the actual numbers can be different from one computer to another computer

Question

```
x = [ 1, 4, 1 ]
y = x
y[1] = 0

# what is x?
```

Question

```
x = [ 1, 4, 1 ]
y = x
y[1] = 0

# what is x?
```

x = [1, 0, 1]

Container Methods - list

Methods like `.sort()` and `.append()` modify the `list` immediately as a `list` is mutable.

Since these methods changed the list immediately, no reason to `return` the final value.

Warning!

This explains why `.sort()` and `.append()` return `None`!
But *not* all list.methods will return `None`!

```
x = [ 4, 1, 2, 3 ]

x.sort() #This is the right way to sort a list.

x = x.sort() #This is the wrong way to sort a list.
```

Question 1

```
x = [ 9, 1, 1, 3 ]
y = x.sort()
z = x.count(1)

# What are the values of x, y, z
    after the above operations?
```

Question 1

```
x = [ 9, 1, 1, 3 ]
y = x.sort()
z = x.count(1)

# What are the values of x, y, z
    after the above operations?
```

Ans:

```
x = [1, 1, 3, 9]
y = None
z = 2          #not None
```

Question 2

```
x = [ 'a', 'b' ]
y = [ 'c', 'd' ]

def add_it( a,b ):
    b.append( a )

add_it( y,x )
```

What is the final value of `x`?

A `[ 'a', 'b', 'c', 'd' ]`

B `[ 'a', 'b' ]`

C `[ 'a', 'b', [ 'c', 'd' ] ]`

D `[ 'c', 'd', [ 'a', 'b' ] ]`

E `Error`

Question 2

```
x = [ 'a', 'b' ]
y = [ 'c', 'd' ]
def add_it( a,b ):
    b.append( a )

add_it( y,x )
```

What is the final value of `x`?

A `[ 'a', 'b', 'c', 'd' ]`

B `[ 'a', 'b' ]`

C `[ 'a', 'b', [ 'c', 'd' ] ]` ⋆

D `[ 'c', 'd', [ 'a', 'b' ] ]`

E `Error`

Question 2

```
x = [ 'a', 'b' ]
y = [ 'c', 'd' ]

def add_it( a,b ):
    b.append( a )

add_it( y,x )
```

What is the final value of `x`?

A `[ 'a', 'b', 'c', 'd' ]`

B `[ 'a', 'b' ]`

C `[ 'a', 'b', [ 'c', 'd' ] ]` ⋆`append` put it inside!

D `[ 'c', 'd', [ 'a', 'b' ] ]`

E `Error`

Question 2 more...

```
x1 = [ 'a', 'b' ]
x2 = x1[ : ]
y1 = [ 'c', 'd' ]
y2 = y1[ : ]

x1.append( y1 )
x2.extend( y2 )
```

Question 2 more...

```
x1 = [ 'a', 'b' ]
x2 = x1[ : ]
y1 = [ 'c', 'd' ]
y2 = y1[ : ]

x1.append( y1 )
x2.extend( y2 )
```

`x1 = [ 'a', 'b', [ 'c', 'd' ] ]` `append` put everything inside!

`x2 = [ 'a', 'b', 'c', 'd' ]` `extend` just put the elements

Question 3

```
x = ( '3', '4' )
y = ( '8', '6' )

def add_it( a,b ):
    b.append( a )

add_it( y,x )
```

What is the final value of `x`?

A `( '3', '4', '8', '6' )`

B `( '3', '4' )`

C `( '3', '4', ( '8', '6' ) )`

D `( '8', '6', ( '3', '4' ) )`

E `Error`

Question 3

```
x = ( '3', '4' )
y = ( '8', '6' )

def add_it( a,b ):
    b.append( a )

add_it( y,x )
```

What is the final value of `x`?

A `( '3', '4', '8', '6' )`

B `( '3', '4' )`

C `( '3', '4', ( '8', '6' ) )`

D `( '8', '6', ( '3', '4' ) )`

E `Error` ⋆

Question 4

```
x = [ 'a','b','c','d' ]
y = x.sort()
z = ( y is x )
```

What is the final value of `z`?

A `True`

B `False`

Question 4

```
x = [ 'a','b','c','d' ]
y = x.sort()
z = ( y is x )
```

What is the final value of `z`?

A `True`

B `False` ⋆

is not ==

Be sure to distinguish:

```
x = [ 'a','b','c','d' ]
y = x
z = x[ : ]

x is y
z is not x
x == y
x == z
```

is not ==

Be sure to distinguish:

```
x = [ 'a','b','c','d' ]
y = x
z = x[ : ]

x is y
z is not x
x == y
x == z
```

The above are all True

Multidimensional Indexing

Nested lists

lec04, we touched on `list`

Just as we can nest control structures in *lec03* (e.g., `for` in a `for` loop), we can nest container values.

```
a = [ [ 1, 2 ], [ 3, 4 ] ]
```

What does this look like to you?

Nested lists

lec04, we touched on `list`

Just as we can nest control structures in *lec03* (e.g., `for` in a `for` loop), we can nest container values.

```
a = [ [ 1, 2 ], [ 3, 4 ] ]
```

What does this look like to you? A matrix or 2D array.

Multidimensional indexing

Access member values of a nested container by coordinates or a group of numbers:

```
a = [ [ 1, 2 ], [ 3, 4 ] ]
a[0]     #?
a[0][0] #?
```

Multidimensional indexing

Access member values of a nested container by coordinates or a group of numbers:

```
a = [ [ 1, 2 ], [ 3, 4 ] ]
a[0]     #?
a[0][0] #?
```

Ans:

a[0] = [1, 2]

a[0][0] = 1

Python orders by `(row, column)`—that is, the first number selects the row and the second selects the column in that row.

Multidimensional indexing

Access member values of a nested container by coordinates or a group of numbers:

```
a = [ [ 1, 2 ], [ 3, 4 ] ]
a[0]     #?
a[0][0] #?
```

Ans:

a[0] = [1, 2]

a[0][0] = 1

Python orders by `(row, column)`—that is, the first number selects the row and the second selects the column in that row.

Side effect: easy to select “rows”, hard to select “columns”!

Example

```
a = [ [1,2,3], [4,5,6], [7,8,9] ]
```

How would you refer to the value 6?

A `a[2][3]`

B `a[1][2]`

C `a[2,3]`

D `a[2][1]`

Example

```
a = [ [1,2,3], [4,5,6], [7,8,9] ]
```

How would you refer to the value 6?

A `a[2][3]`

B `a[1][2]` ⋆

C `a[2,3]`

D `a[2][1]`

Example

```
a = [ [1,2,3], [4,5,6], [7,8,9] ]
```

How would you refer to the value 6?

A `a[2][3]`

B `a[1][2]` ⋆

C `a[2,3]`

D `a[2][1]`

or `a[-2][-1]` ⋆

Dictionaries

Container data type

How do we index an element of a `list`?

Container data type

How do we index an element of a `list`? integers!

From last 2 lectures, we see `list`s and `tuple`s are *ordered* containers (i.e., the items are in the order you typed in), so using `int`s to look up each element makes sense.

e.g., `x[0]` gives you the first element in `x`

Container data type

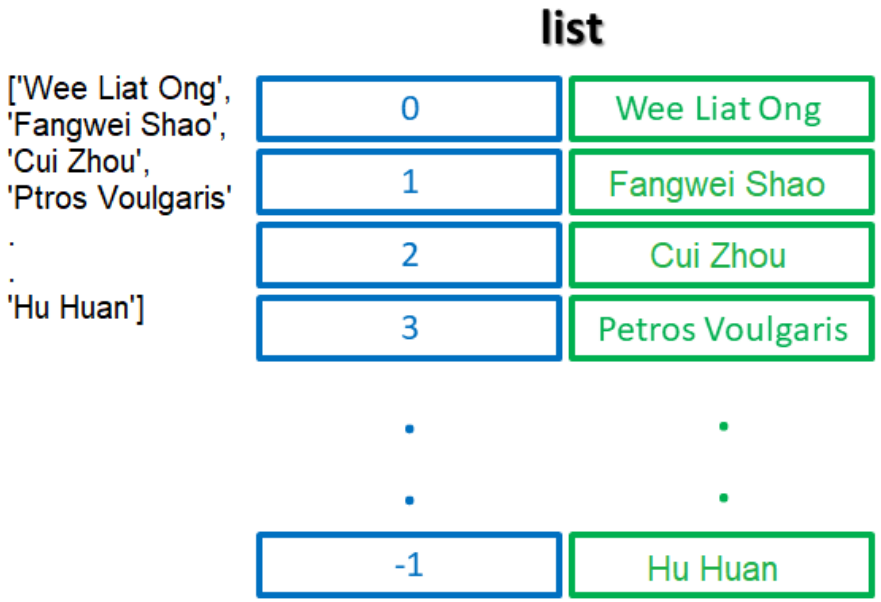
How do we index an element of a `list`? integers!

From last 2 lectures, we see `list`s and `tuple`s are *ordered* containers (i.e., the items are in the order you typed in), so using `int`s to look up each element makes sense.

e.g., `x[0]` gives you the first element in `x`

What other ”stuff” can be used to look up for elements in a container?

list data type



dict data type

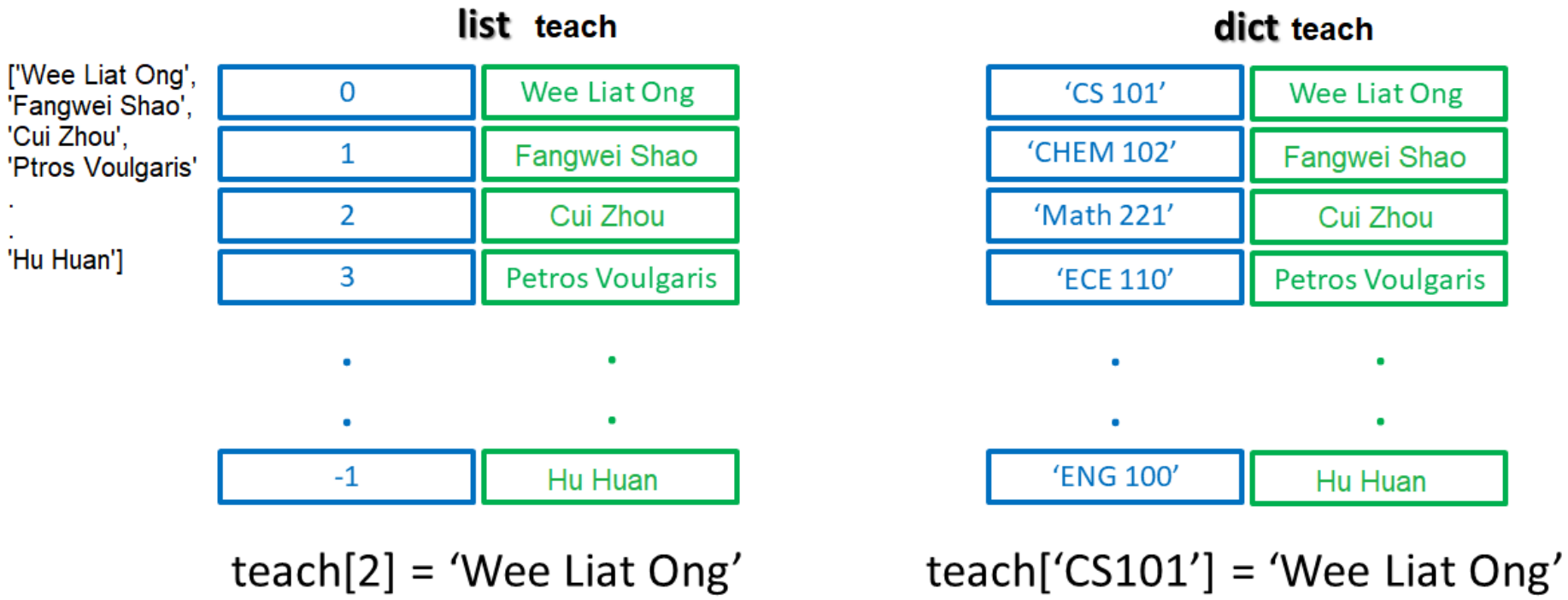
dict data type

dict data type

The `dict` accesses data by *key* (*unordered* container before Py3.6).

Easy to think of as dictionary, but can use other data types besides strings.

This container maps *keys* to *values*.



dict literals

We create a `dict` as follows:

- variable name =
- opening brace `{`
- `key : value` pairs, separated by commas
- closing brace `}`

`Keys` can be any *immutable* type: `int`, `float`, `str`, `tuple`, `boolean`

A same `key` can only appear once else your earlier `value` will be over-written

dict literals

```
model = {
   'iPhone XS': 'Apple',
   'Mate RS': 'Huawei',
   'Find X': 'Oppo',
   'Mix 2s': 'Xiaomi'
}
```

dict literals

```
model = {
   'iPhone XS': 'Apple',
   'Mate RS': 'Huawei',
   'Find X': 'Oppo',
   'Mix 2s': 'Xiaomi'
}

variable = {
     key : value,
           .
           .
           .
}
```

dict data type

Another way is to do this:

```
model = {}
model['iPhone XS'] = 'Apple'
model['Mate RS' ] = 'Huawei'
model['Find X' ] = 'Oppo'
model['Mix 2s' ] = 'Xiaomi'
```

list, tuple, dict

Creation:

`list`: `x = [1, 2, 4]`

`tuple`: `y = (1, 2, 4)` *often you can drop the ( )

`dict`: `z ={'One' : 1, 2.0 : 2, 3 : 'Three'}`

list, tuple, dict

Creation:

`list`: `x = [1, 2, 4]`

`tuple`: `y = (1, 2, 4)` *often you can drop the ( )

`dict`: `z ={'One' : 1, 2.0 : 2, 3 : 'Three'}`

Access:

All use [ ].

```
x[ 0 ] = 1
y[ 0 ] = 1
z['One'] = 1
```

dict common methods

```
d = {1:'1', Two':2, 3.0:3.0 4:'Four'}
```

`d.clear( )` empties dictionary d of all key-value pairs

`d.get( key )` gets the value of a `key` from d

`d.items( )` returns a list of tuples containing the key-value pairs in d

`d.keys( )` returns a list of all keys in d

`d.values( )` returns a list of all values in d

dict operations & methods

```
d = { 'one':1, 'two':2, 'three':3, 4.0:4 }

print( d['one'] )
```

dict operations & methods

```
d = { 'one':1, 'two':2, 'three':3, 4.0:4 }

print( d['one'] )
```

Ans: `1`

dict operations & methods

```
d = { 'one':1, 'two':2, 'three':3, 4.0:4 }

print( d['one'] )
```

Ans: 1

```
d[ 5 ] = 'five'   # map int 5 to str 'five'
print(d)
```

dict operations & methods

```
d = { 'one':1, 'two':2, 'three':3, 4.0:4 }

print( d['one'] )
```

Ans: `1`

```
d[ 5 ] = 'five'   # map int 5 to str 'five'
print(d)
```

Ans: `{'one':1, 'two':2, 'three':3, 4.0:4, 5:'five'}`

dict operations & methods

```
d = { 'one':1, 'two':2, 'three':3, 4.0:4 }

print( d['one'] )
```

Ans: 1

```
d[ 5 ] = 'five'   # map int 5 to str 'five'
print(d)
```

Ans: {'one':1, 'two':2, 'three':3, 4.0:4, 5:'five'}

```
del d[ 4.0 ]
print(d)
```

dict operations & methods

```
d = { 'one':1, 'two':2, 'three':3, 4.0:4 }

print( d['one'] )
```

Ans: `1`

```
d[ 5 ] = 'five'   # map int 5 to str 'five'
print(d)
```

Ans: `{'one':1, 'two':2, 'three':3, 4.0:4, 5:'five'}`

```
del d[ 4.0 ]
print(d)
```

Ans: `{'one':1, 'two':2, 'three':3, 5:'five'}`

dict operations & methods

```
d = { 'one':1, 'two':2, 'three':3, 4.0:4 }

print( d['one'] )
```

Ans: 1

```
d[ 5 ] = 'five'   # map int 5 to str 'five'
print(d)
```

Ans: {'one':1, 'two':2, 'three':3, 4.0:4, 5:'five'}

```
del d[ 4.0 ]
print(d)
```

Ans: {'one':1, 'two':2, 'three':3, 5:'five'}

```
'six' in d
```

dict operations & methods

```
d = { 'one':1, 'two':2, 'three':3, 4.0:4 }

print( d['one'] )
```

Ans: 1

```
d[ 5 ] = 'five'   # map int 5 to str 'five'
print(d)
```

Ans: {'one':1, 'two':2, 'three':3, 4.0:4, 5:'five'}

```
del d[ 4.0 ]
print(d)
```

Ans: {'one':1, 'two':2, 'three':3, 5:'five'}

```
'six' in d
```

Ans: False

dict operations & methods

```
d = { 'one':1, 'two':2, 'three':3, 5:'five' }

for key in d:     # no guarantee on order b4 Py3.6
    print( key, d[key] )
d.keys()
d.values()
```

dict operations & methods

```
d = { 'one':1, 'two':2, 'three':3, 5:'five' }

for key in d:     # no guarantee on order b4 Py3.6
    print( key, d[key] )
d.keys()
d.values()
```

Ans:

```
one 1
two 2
three 3
5 five
dict_keys(['one', 'two', 'three', 5])
dict_values([1, 2, 3, 'five'])
```

Question 1

```
d = { 'a':2, 'c':3, 'b':1 }
x = d[ 'a' ] + d[ 'c' ]
```

What is the final value of `x`?

A `4`

B `'ac'`

C `'5'`

D `5`

Question 1

```
d = { 'a':2, 'c':3, 'b':1 }
```

Question 1

```
d = { 'a':2, 'c':3, 'b':1 }
x = d[ 'a' ] + d[ 'c' ]
```

Question 1

```
d = { 'a':2, 'c':3, 'b':1 }
x = d[ 'a' ] + d[ 'c' ]
x = 2 + d[ 'c' ]
```

Question 1

```
d = { 'a':2, 'c':3, 'b':1 }
x = d[ 'a' ] + d[ 'c' ]
x = 2 + d[ 'c' ]
x = 2 + 3
```

Question 1

```
d = { 'a':2, 'c':3, 'b':1 }
x = d[ 'a' ] + d[ 'c' ]
x = 2 + d[ 'c' ]
x = 2 + 3
x = 5
```

Question 1

```
d = { 'a':2, 'c':3, 'b':1 }
x = d[ 'a' ] + d[ 'c' ]
```

What is the final value of `x`?

A `4`

B `'ac'`

C `'5'`

D `5` ⋆

Question 2

```
d = { 'a':'2', 'c':'3', 'b':'1' }
x = d[ 'a' ] + d[ 'c' ]
```

What is the final value of `x`?

A `23`

B `5`

C `'23'`

D `'5'`

Question 2

```
d = { 'a':'2', 'c':'3', 'b':'1' }
x = d[ 'a' ] + d[ 'c' ]
```

What is the final value of `x`?

A `23`

B `5`

C `'23'` ⋆

D `'5'`

Question 3

```
d = { }
words = [ 'red', 'orange', 'yellow' ]
for word in words:
    d[ word ] = words.index( word )
```

What is the final value of `d`?

A `{ 'red':3, 'orange':6, 'yellow':6 }`

B `{ 'red':0, 'orange':2, 'yellow':2 }`

C `None`

D `{'orange': 1, 'red': 0, 'yellow': 2}`

E `{'red': 0, 'orange': 1, 'yellow': 2}`

Question 3

```
d = { }
words = [ 'red', 'orange', 'yellow' ]
for word in words:
    d[ word ] = words.index( word )
```

What is the final value of `d`?

A `{ 'red':3, 'orange':6, 'yellow':6 }`

B `{ 'red':0, 'orange':2, 'yellow':2 }`

C `None`

D `{'orange': 1, 'red': 0, 'yellow': 2}` ⋆maybe correct after py2 and py3

E `{'red': 0, 'orange': 1, 'yellow': 2}` ⋆⋆after py3.6

Question 3

```
d = { }
words = [ 'red', 'orange', 'yellow' ]
for word in words:
    d[ word ] = words.index( word )

for word in ['red', 'orange', 'yellow']:
```

Question 3

```
d = { }
words = [ 'red', 'orange', 'yellow' ]
for word in words:
    d[ word ] = words.index( word )

for word in ['red', 'orange', 'yellow']:
# here word = 'red'
    d[ 'red' ] = words.index( 'red' )
```

Question 3

```
d = { }
words = [ 'red', 'orange', 'yellow' ]
for word in words:
    d[ word ] = words.index( word )

for word in ['red', 'orange', 'yellow']:
```

here word = 'red'

```
    d[ 'red' ] = words.index( 'red' )

    d[ 'red' ] = 0
```

d = {'red' : 0}

Question 3

```
for word in ['red', 'orange', 'yellow']:
```

Question 3

```
for word in ['red', 'orange', 'yellow']:
```

here word = 'orange'

```
    d[ 'orange' ] = words.index( 'orange' )
    d[ 'orange' ] = 1
```

d = {'red' : 0, 'orange' : 1}

Question 3

```
for word in ['red', 'orange', 'yellow']:
```

here word = 'orange'

```
    d[ 'orange' ] = words.index( 'orange' )
    d[ 'orange' ] = 1
```

d = {'red' : 0, 'orange' : 1}

```
for word in ['red', 'orange', 'yellow']:
```

here word = 'yellow'

Question 3

```
for word in ['red', 'orange', 'yellow']:
```

here word = 'orange'

```
    d[ 'orange' ] = words.index( 'orange' )
    d[ 'orange' ] = 1
```

d = {'red' : 0, 'orange' : 1}

```
for word in ['red', 'orange', 'yellow']:
```

here word = 'yellow'

```
    d[ 'yellow' ] = words.index( 'yellow' )
    d[ 'yellow' ] = 2
```

d = {'red' : 0, 'orange' : 1, 'yellow' : 2}

Question 3

```
d = { }
words = [ 'red', 'orange', 'yellow' ]
for word in words:
    d[ word ] = words.index( word )
```

What is the final value of `d`?

A `{ 'red':3, 'orange':6, 'yellow':6 }`

B `{ 'red':0, 'orange':2, 'yellow':2 }`

C `None`

D `{'orange': 1, 'red': 0, 'yellow': 2}` ⋆maybe correct after py2 and py3

E `{'red': 0, 'orange': 1, 'yellow': 2}` ⋆⋆after py3.6

dict application 1 - Encode

Dictionaries can encode/decode data, or translate from one representation to another.

dict application 1 - Encode

Dictionaries can encode/decode data, or translate from one representation to another.

```
x = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
y = 'BCDEFGHIJKLMNOPQRSTUVWXYZA'
e = { }
for i in range( len( x ) ):
    e[ x[ i ] ] = y[ i ]

encoded = ''
for c in 'HELLO':
    encoded += e[ c ]
```

What is the value of `encoded`?

dict application 1 - Encode

```
x = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
y = 'BCDEFGHIJKLMNOPQRSTUVWXYZA'
e = { }
for i in range( len( x ) ):
# i = 0
    e[x[i]] = y[i]
    e[x[0]] = y[0]
    e['A'] = 'B'
```

dict application 1 - Encode

```
x = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
y = 'BCDEFGHIJKLMNOPQRSTUVWXYZA'
e = { }
for i in range( len( x ) ):
# i = 0
    e[x[i]] = y[i]
    e[x[0]] = y[0]
    e['A'] = 'B'

for i in range( len( x ) ):
# i = 1
    e[x[i]] = y[i]
    e[x[1]] = y[1]
    e['B'] = 'C'
...
```

dict application 1 - Encode

```
x = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
y = 'BCDEFGHIJKLMNOPQRSTUVWXYZA'
e = { }
for i in range( len( x ) ):
# i = 0
    e[x[i]] = y[i]
    e[x[0]] = y[0]
    e['A'] = 'B'

for i in range( len( x ) ):
# i = 1
    e[x[i]] = y[i]
    e[x[1]] = y[1]
    e['B'] = 'C'
...

e = {'A':'B', 'B':'C', 'C':'D' ... 'Z':'A'}
```

dict application 1 - Encode

```
e = {'A':'B', 'B':'C', 'C':'D' ... 'Z':'A'}

encoded = ''
for c in 'HELLO':
# c = 'H'
    encoded += e[c]
    encoded += e['H']
    encoded = '' + 'I'
```

dict application 1 - Encode

```
e = {'A':'B', 'B':'C', 'C':'D' ... 'Z':'A'}

encoded = ''
for c in 'HELLO':
# c = 'H'
    encoded += e[c]
    encoded += e['H']
    encoded = '' + 'I'

for c in 'HELLO':
# c = 'E'
    encoded += e[c]
    encoded += e['E']
    encoded = 'I' + 'F'
...
```

dict application 1 - Encode

```
e = {'A':'B', 'B':'C', 'C':'D' ... 'Z':'A'}

encoded = ''
for c in 'HELLO':
# c = 'H'
    encoded += e[c]
    encoded += e['H']
    encoded = '' + 'I'

for c in 'HELLO':
# c = 'E'
    encoded += e[c]
    encoded += e['E']
    encoded = 'I' + 'F'
...
```

Ans: `encoded = 'IFMMP'`

dict application 2 - Decode

Dictionaries can encode/decode data, or translate from one representation to another.

```
x = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
y = 'BCDEFGHIJKLMNOPQRSTUVWXYZA'
e = { }
for i in range( len( x ) ):
    e[ x[ i ] ] = y[ i ]
encoded = ''
for c in 'HELLO':
    encoded += e[ c ]
```

Ans: `encoded = 'IFMMP'`

dict application 2 - Decode

Dictionaries can encode/decode data, or translate from one representation to another.

```
x = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
y = 'BCDEFGHIJKLMNOPQRSTUVWXYZA'
e = { }
for i in range( len( x ) ):
    e[ x[ i ] ] = y[ i ]
encoded = ''
for c in 'HELLO':
    encoded += e[ c ]
```

Ans: `encoded = 'IFMMP'`

How would you reverse (decode) this?

dict application 2 - Decode

Dictionaries can encode/decode data, or translate from one representation to another.

```
x = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
y = 'BCDEFGHIJKLMNOPQRSTUVWXYZA'
e = { }
for i in range( len( x ) ):
    e[ x[ i ] ] = y[ i ]
encoded = ''
for c in 'HELLO':
    encoded += e[ c ]
```

Ans: `encoded = 'IFMMP'`

How would you reverse (decode) this?

`'IFMMP'` back to `'HELLO'`?

dict application 2 - Decode

```
x = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
y = 'BCDEFGHIJKLMNOPQRSTUVWXYZA'
d = { }
for i in range( len( x ) ):
    d [ y[ i ] ] = x[ i ]  #########
decoded = ''
for c in encoded:
    decoded += d[ c ]
```

dict application 3

Dictionaries can also function as accumulators/counters.

```
# count the vowels
vowels = 'aeiou'
text = 'Today is an exciting day because
          we are going to know our exam01 results'
```

dict application 3

Dictionaries can also function as accumulators/counters.

```
# count the vowels
vowels = 'aeiou'
text = 'Today is an exciting day because
          we are going to know our exam01 results'

text = text.lower()
d = { 'a':0, 'e':0, 'i':0, 'o':0, 'u':0 }
for c in text:
    if c in vowels:
        d[ c ] += 1
```

dict application 4

We can link data from different `dict` based on a common field.

```
zipcode = { 'Jiajia': 314400,
            'Zhiling': 310058,
            'Tony': 63103 }
city = { 314400: 'Haining',
         310058: 'Hangzhou',
         63103: 'St. Louis' }

for name in zipcode:
    print( name,city[ zipcode[ name ] ] )
```

Summary

Summary

A. Nested `list` and Access using multiple level indexing
B. `dict` a mutable type
C. Create using { key:value } and access using [key]
D. methods for `dict`