Python Programming Language Foundation

Session 2

Session overview

- Data types
- Operators
- Conditions
- Loops

Data types

Data types

numbers

dictionary

tuple

set

list

string

bool

https://docs.python.org/3/library/datatypes.html

Data types

Data types

Immutable

Mutable

In Python everything is an object

https://jakevdp.github.io/WhirlwindTourOfPython/03-semantics-variables.html#Everything-Is-an-Object

Variables

Variable is a name attached to a particular object

https://realpython.com/python-variables

Strings

String is immutable collection of symbols

```
some_string = 'don\'t'
```

```
some_string = "don't"
```

Immutability mechanism

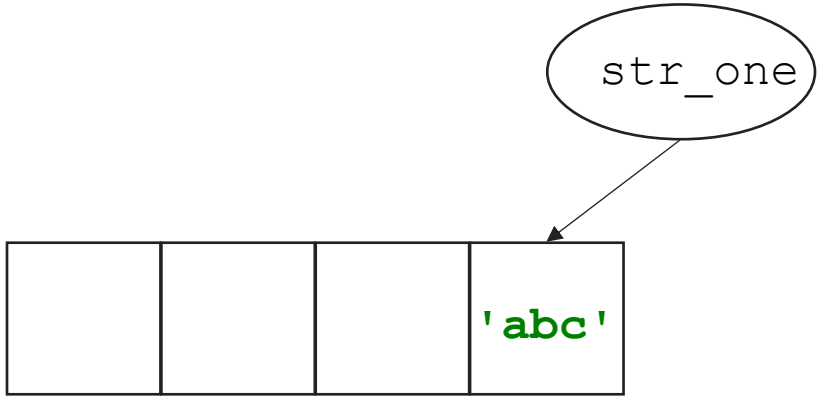
```
>>> str_one = 'abc'

>>> str_two = 'def'

>>> str_one = str_one + str_two

>>> print(str_one)
'abcdef'
```

Immutability mechanism

Code line:

```
>>> str_one = 'abc'
```

Memory:

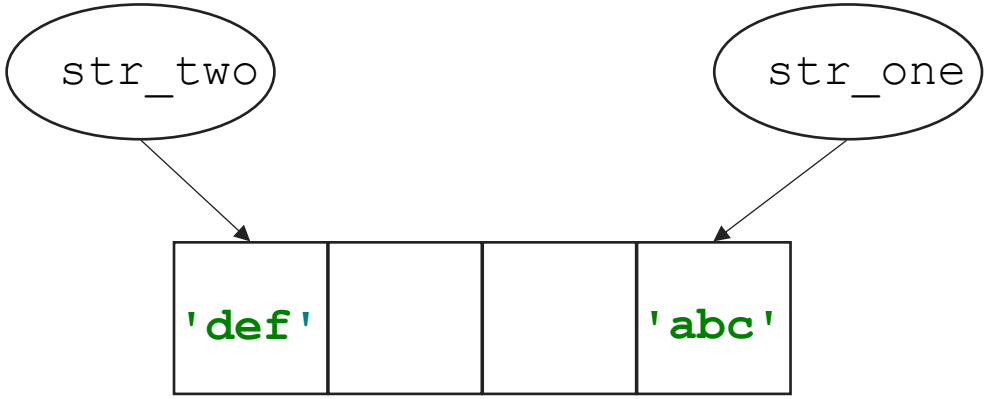


Immutability mechanism

Code line:

```
>>> str_two = 'def'
```

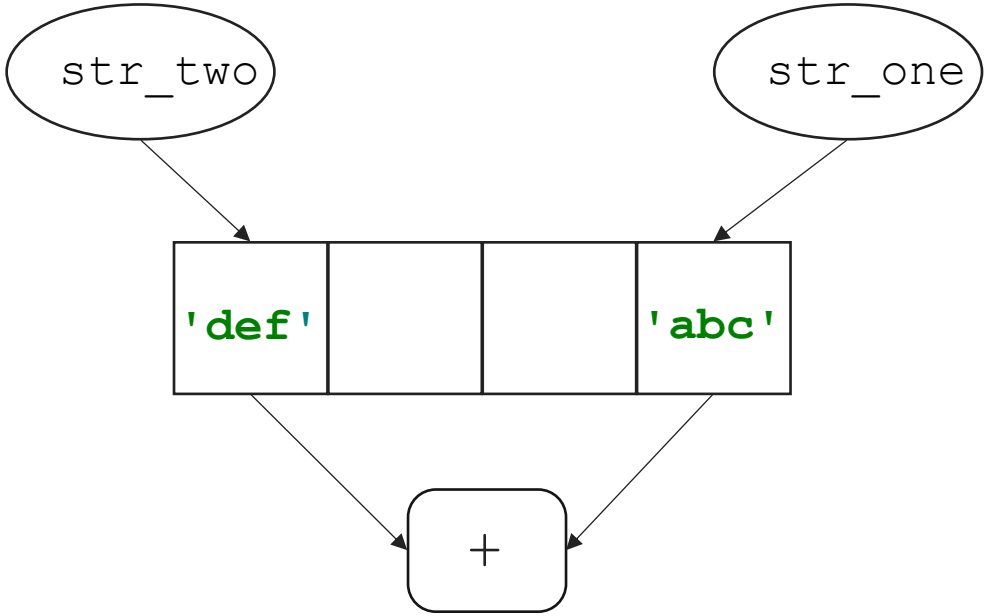
Memory:



Immutability mechanism

Code line:

Memory:

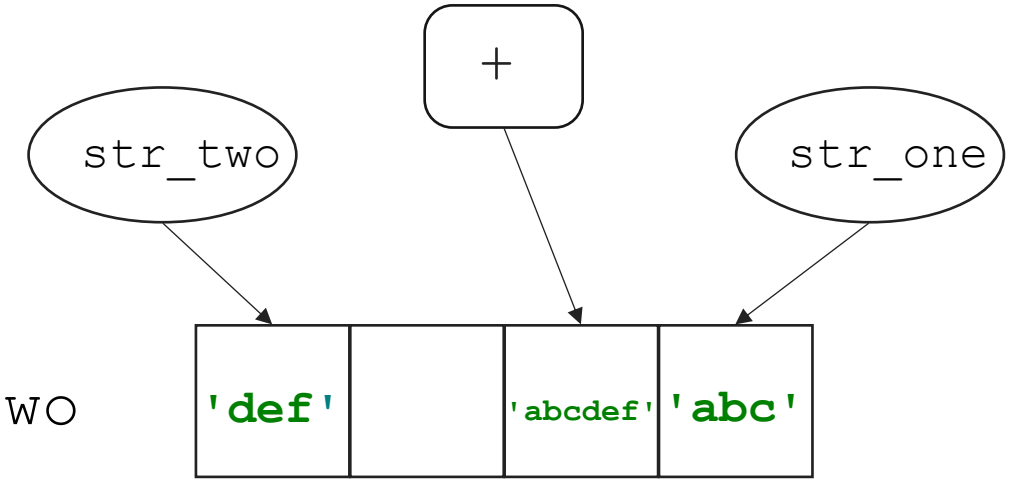


Immutability mechanism

Code line:

```
>>> str_one = str_one + str_two
```

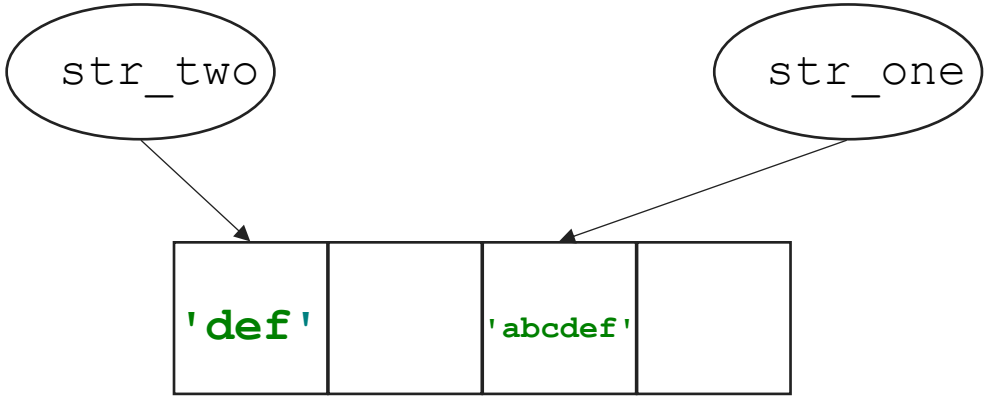
Memory:



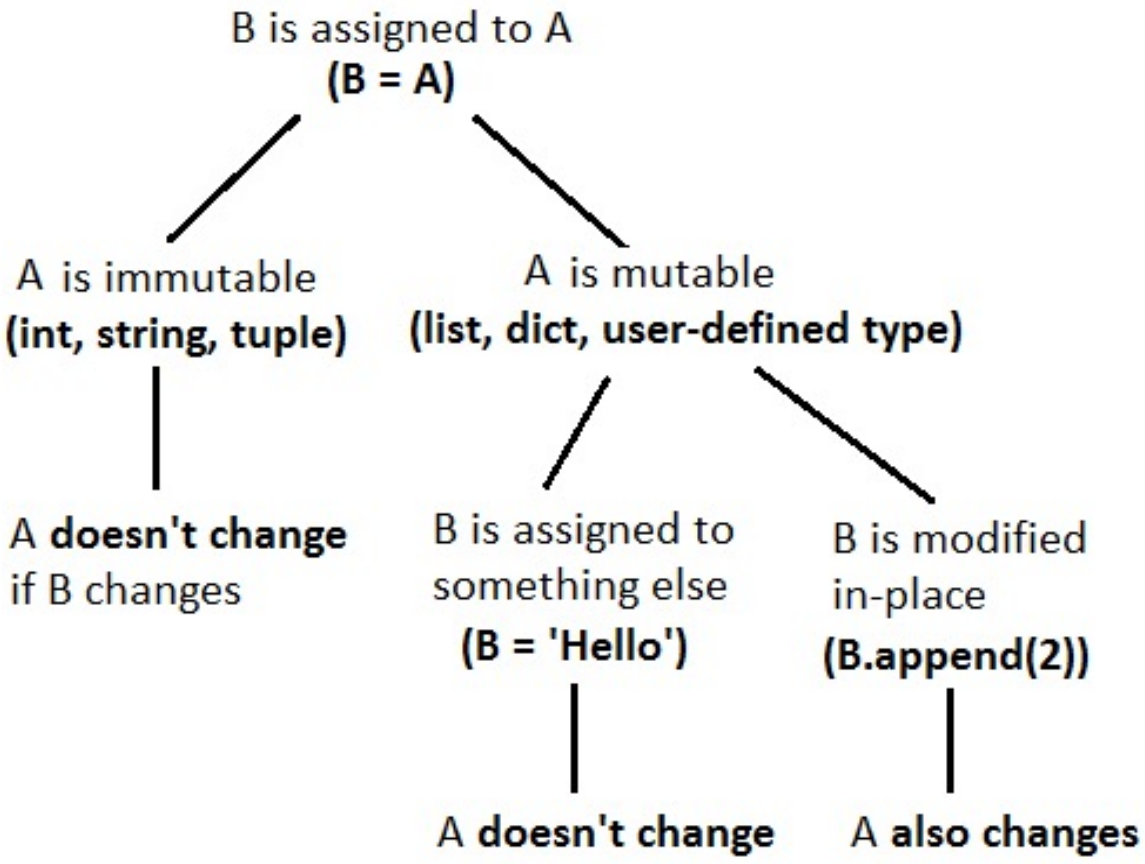
Immutability mechanism

Code line:

Memory:



Immutable vs Mutable



https://medium.com/@704/mutable-or-immutable-that-is-the-question-a-work-in-progress-fc7f658b340a

Accessing string elements

```
some_string[i]
```

Slices

```
some_string[start:end:step]
```

String formatting

% Operator

str.format

f-Strings

https://realpython.com/python-string-formatting/

Different string literals

\n

\t

r'\nstring'

b'string'

https://docs.python.org/3/reference/lexical_analysis.html#string-and-bytes-literals

String operators

Expression	Result	Description
`len('abc')`	`3`	Length
`'abc'+'cde'`	`'abccde'`	Concatenation
`"O"*3`	`"OOO"`	Reiteration
`'a' in 'abc'`	`True`	Affiliation
`for x in 'abc': print(x)`	`a b c`	Iterating

String methods

- `len(string)`
- `endswith(suffix, beg=0, end=len(string)), startswith(str, beg=0,end=len(string))`
- `rstrip(), lstrip(), strip([chars])`
- `find(str, beg=0 end=len(string)), count(str, beg= 0,end=len(string))`
- `join(seq), split(str="", num=string.count(str))`
- `replace(old, new [, max])`
- `upper(), lower()`

Numbers

Numbers types

int → `123`

float → `123.0`

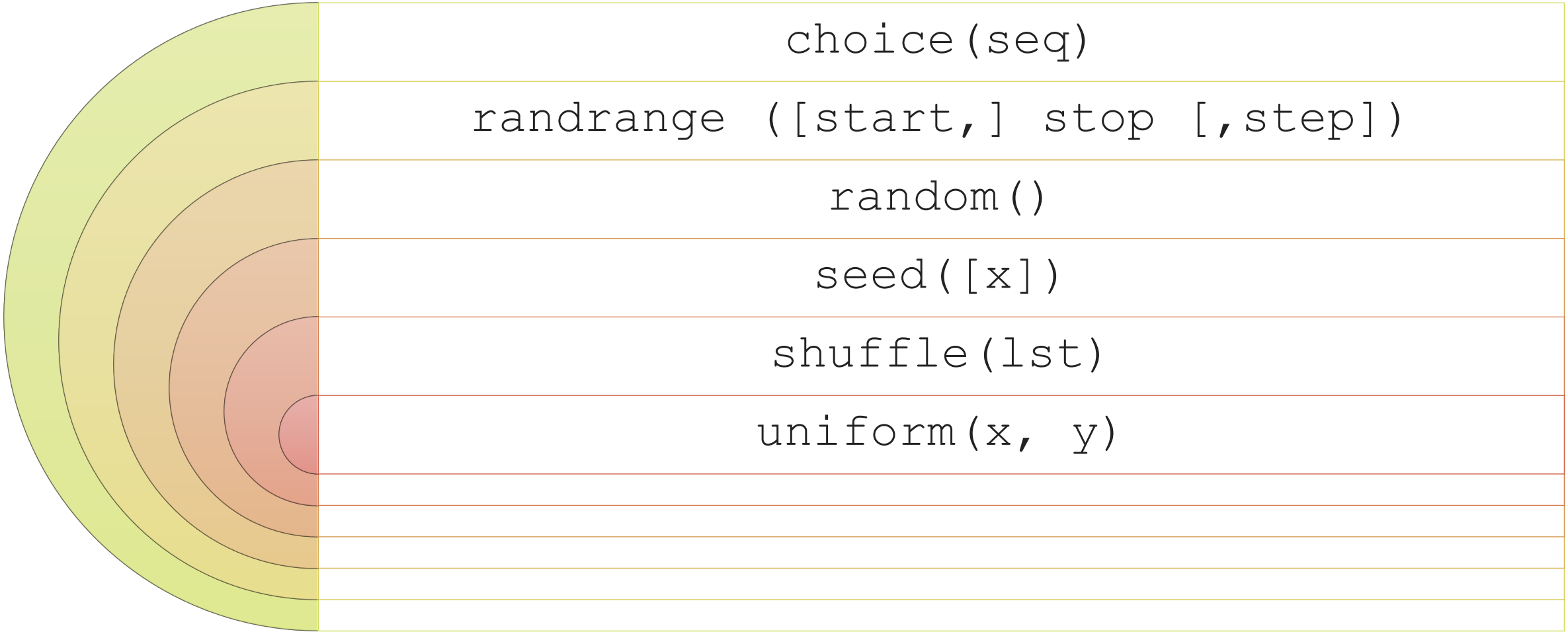
Numbers types

```
abs(x)
exp(x)
log(x)
log10(x)
pow(x, y)
round(x [,n])
sqrt(x)
```

Trigonometry (math module)

`acos(x)`	Arc cosine
`asin(x)`	Arc sine
`atan(x)`	Arc tangent
`atan2(y, x)`	atan(y / x)
`cos(x)`	Cosine
`hypot(x, y)`	Euclidean distance: sqrt(x**2 + y**2)
`sin(x)`	Sine
`tan(x)`	Tangent
`degrees(x)`	Radians to degrees
`radians(x)`	Degrees to radians

Random numbers generation (random module)



Lists & Tuples

Represented as a collection of data

List VS Tuple

List is mutable

Tuple is immutable

```
some_list = [1, 2, 'str']

some_tuple = (1, 2, 'str')
# or 1, 2, 3
```

Accessing list/tuple elements

some_list[i]

some_tuple[i]

```
some_list[i] = 2  # OK

some_tuple[i] = 2  # Error
```

```
del some_list[i]  # OK

del some_tuple[i]  # Error
```

Slices

```
some_list[start:end:step]
```

```
some_tuple[start:end:step]
```

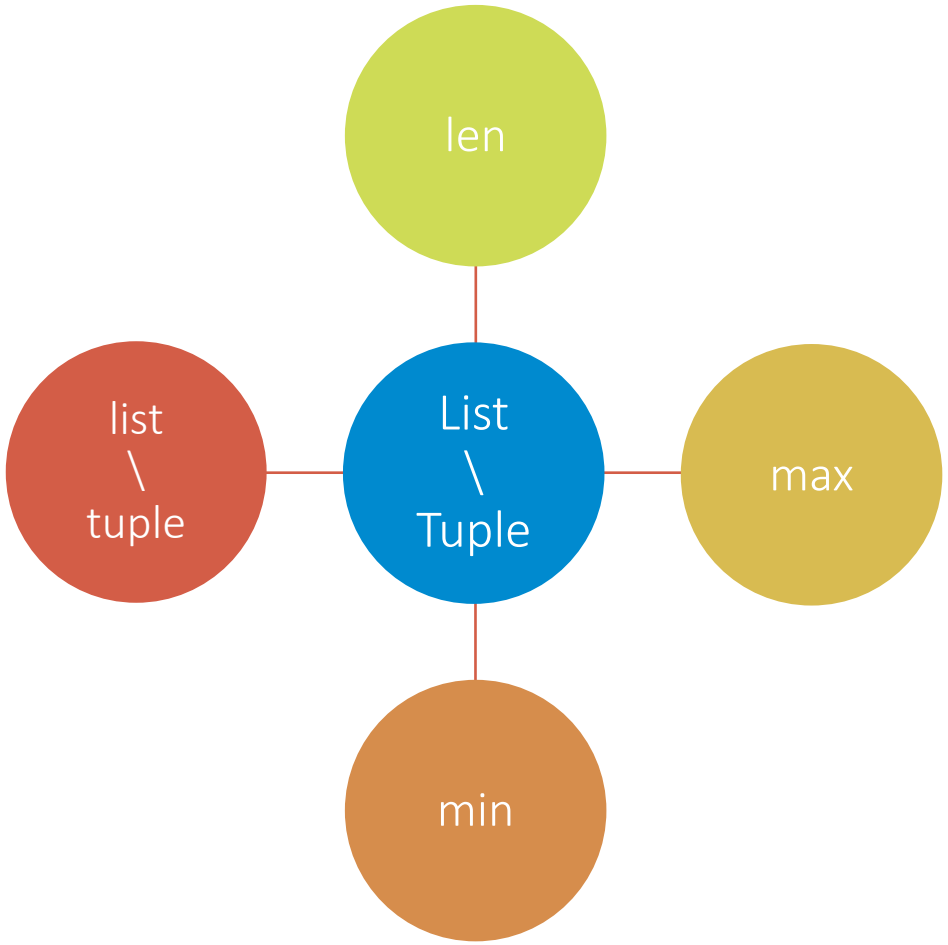
List operators

Expression	Result	Description
`len([1, 2, 3])`	`3`	Length
`[1,2,3]+[4,5,6]`	`[1,2,3,4,5,6]`	Concatenation
`["O"]*3`	`["O", "O", "O", ]`	Reiteration
`3 in [1,2,3]`	`True`	Affiliation
`for x in [1,2,3]:` `  print(x)`	`1 2 3`	Iterating

Tuple operators

Expression	Result	Description
`len((1, 2, 3))`	`3`	Length
`(1,2,3)+(4,5,6)`	`(1,2,3,4,5,6)`	Concatenation
`("O")*3`	`("O", "O", "O", )`	Reiteration
`3 in (1,2,3)`	`True`	Affiliation
`for x in (1,2,3):` `  print(x)`	`1 2 3`	Iterating

Tuple operators



List methods

list	`append(obj)`
	`count(value)`
	`extend(iterable)`
	`index(value, [start, [stop]])`
	`insert(index, obj)`
	`pop([index = -1])`
	`remove(obj)`
	`reverse()`
	`sort(key=None, reverse=False)`

Tuple methods

tuple

~~`append(obj)`~~
`count(value)`
~~`extend(iterable)`~~
`index(value, [start, [stop]])`
~~`insert(index, obj)`~~
~~`pop([index = -1])`~~
~~`remove(obj)`~~
~~`reverse()`~~
~~`sort(key=None, reverse=False)`~~

Bool

Bool

True

False

Everything can be converted to **bool**

NoneType

None

None

Sets

Unordered collection of distinct immutable objects

```
some_set = {1, 2, 3, 'string'}
```

Set operators

Expression	Result	Description
`len({1, 2, 3})`	`3`	Length
`{1,2,3,5,8,13} | {2,3,5,7,11,13}`	`{1, 2, 3, 5, 7, 8, 11, 13}`	Union
`{1,2,3,5,8,13} & {2,3,5,7,11,13}`	`{2, 3, 5, 13}`	Intersection
`{1,2,3,5,8,13} - {2,3,5,7,11,13}`	`{8, 1}`	Difference
`{1,2,3,5,8,13} ^ {2,3,5,7,11,13}`	`{1, 7, 8, 11}`	Symmetric Difference
`3 in {1,2,3}`	`True`	Affiliation
`for x in {1,2,3, 'string', 4}:` `  print(x)`	`1 2 3 4 string`	Iterating

Set methods

set	`clear()`
	`copy()`
	`difference(set, [set1, …])/symmetric_difference(set)`
	`discard(item) / remove(item)`
	`intersection(set, [set1, ...])`
	`isdisjoint(set)`
	`issubset(set)`
	`issuperset(set)`
	`union(set)`

Items

```
special_set = {{1, 2, 3}, {4, 5}}

# TypeError: unhashable type: 'set'
```

Dictionaries

Represented as a collection of key-value pairs where each key-value pair maps the key to its associated value

Features

- Fast access by key
- Fast search by key
- No search by value
- Dictionary is a hash-table

http://thepythoncorner.com/dev/hash-tables-understanding-dictionaries

Keys must be hashable objects

```
some_dict = {'one': 1, 2: 'two'}
```

Accessing dict values

```
some_dict[key]
```

Dict operators

Expression	Result	Description
`len({'one': 1, 'two': 2, 'three': 3})`	`3`	Length
`{'one': 1, 'two': 3} | {'one': 4}`	`{'one': 4, 'two': 3}`	Union
`'three' in {'one': 1, 'two': 2, 'three': 3}`	`True`	Affiliation
`for x in {'one': 1, 'two': 2}:` `  print(x)`	`one two`	Iterating
`for x in {'one': 1, 'two': 2}.values():` `  print(x)`	`1 2`	
`for x, y in {'one': 1}.items():` `  print(x, ':' ,y)`	`one : 1`	

Dict methods

dict

`clear()`
`copy()`
`fromkeys(iterable[,value])`
`get(key[,defaut])`
`items() / keys() / values()`
`pop(key[,default])`
`popitem()`
`setdefault(key[,default])`
`update(dict)`

Hashing

Hashing usage

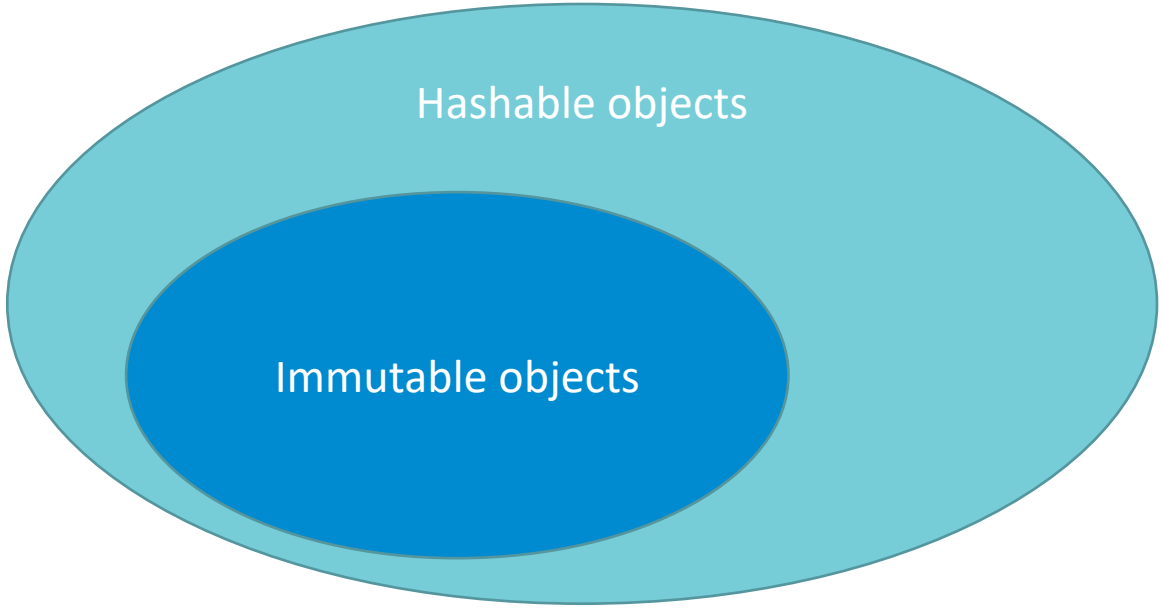
- Store passwords
- Detecting duplicates
- Checksum
- Identify data
- Index data

Hashing

- **Hashing** is the process of translating a given data into a *number*.
- A **hash** is a numeric value of a fixed length that *uniquely identifies data*.
- A **hash function** is used to substitute data with a generated *hash code*.

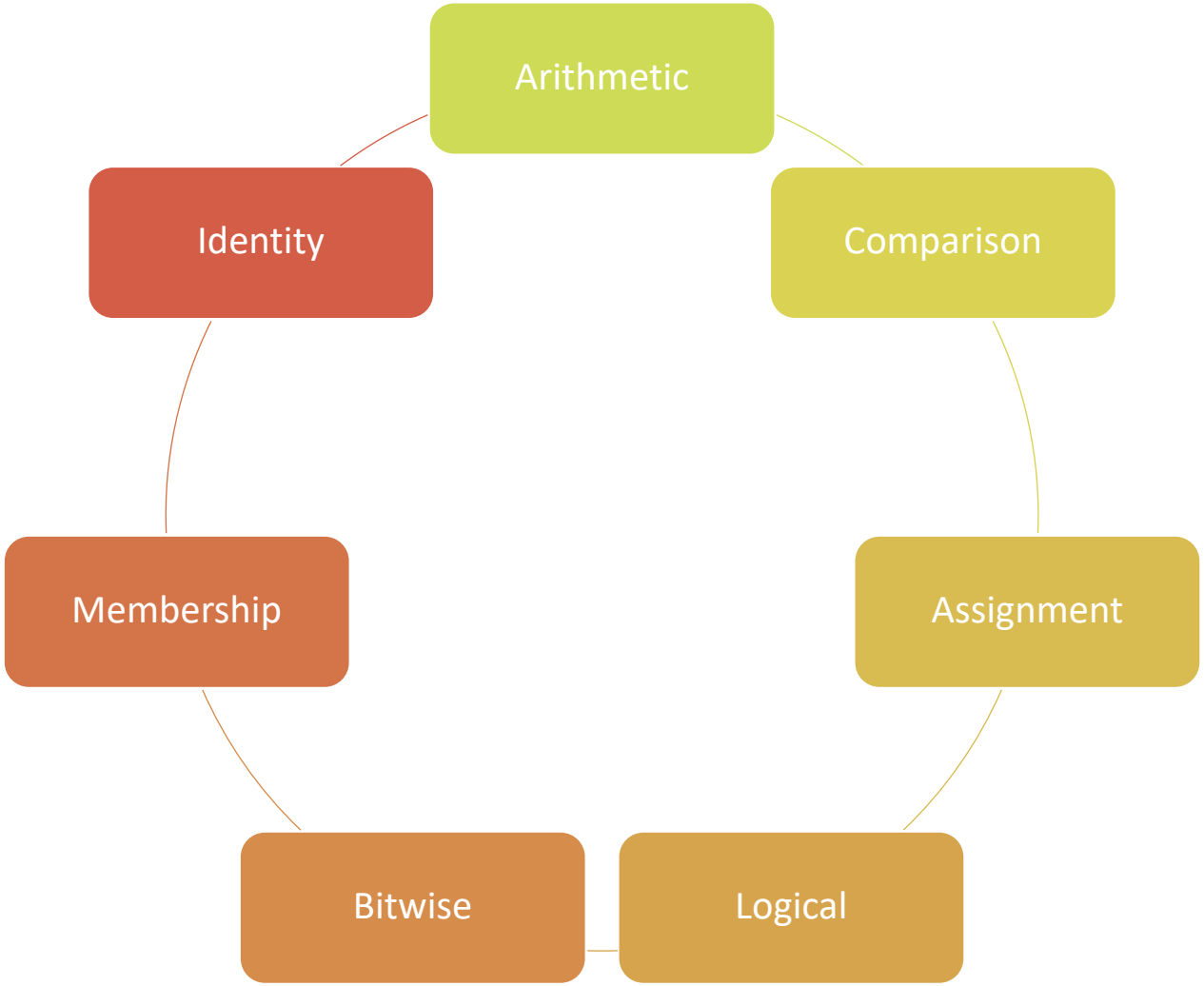
Hashable objects in Python

```
>>>hash(5)
5

>>>hash(5.2)
461168601842739205

>>>hash('string')
4282184674599114870

>>>hash((1, 2, 3))
2528502973977326415

>>>hash(MyOwnCustomObject())
-9223371895723587417

>>>hash([1, 2, 3])
TypeError: unhashable type:
'list'
```



Operators

Operator types



Arithmetic operators

+	• Addition	`print(5 + 2)  # 7`
-	• Subtraction	`print(5 - 2)  # 3`
*	• Multiplication	`print(5 * 2)  # 10`
/	• Division	`print(10 / 4)  # 2.5`
%	• Modulus	`print(5 % 2)  # 1`
**	• Exponent	`print(5 ** 2)  # 25`
//	• Floor division	`print(10 // 4)  # 2`

Comparison (relational) operators

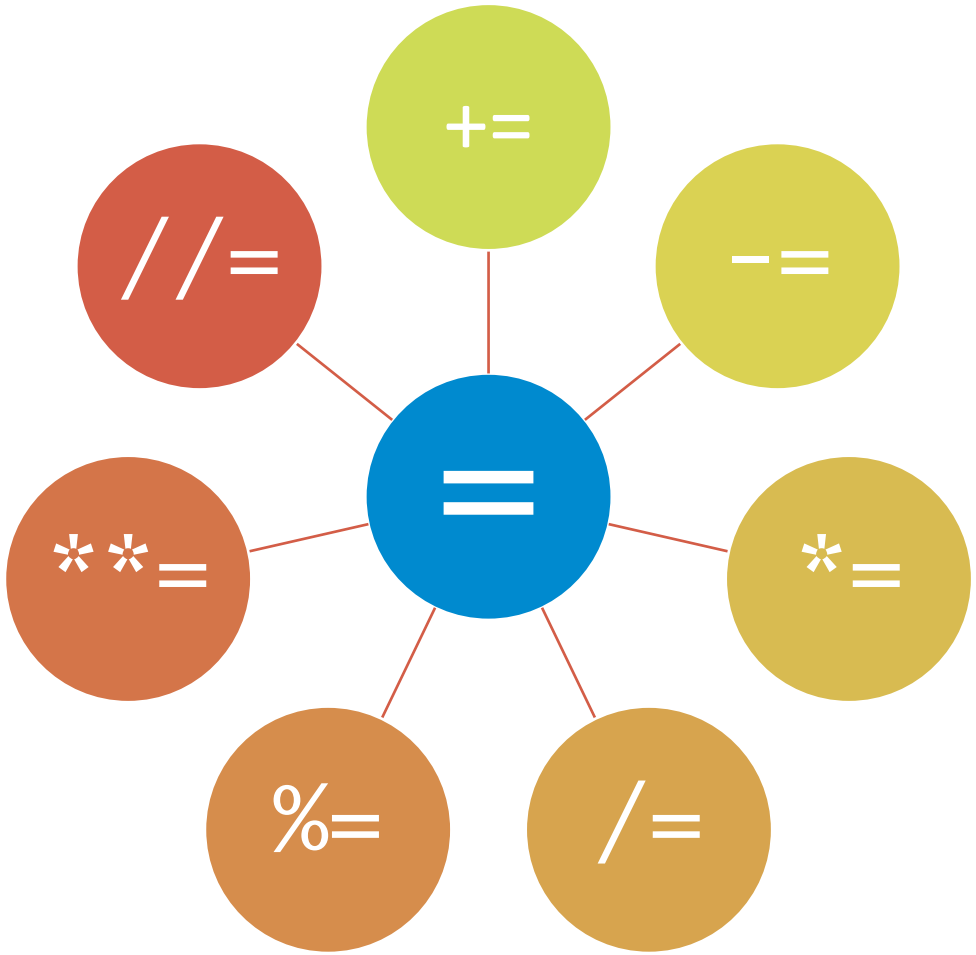
!=	>	<
>=	<=	==

Comparison (relational) operators | Examples

```
>>> print(5 != 2)
True

>>> print(5 >= 5)
True

>>> print(5 > 5)
False
```

```
>>> print(5 < 6)
True

>>> print(5 <= 6)
True

>>> print(5 == 5)
True
```

Assignment operators



Bitwise operators

&	• Bitwise AND
|	• Bitwise OR
^	• Bitwise XOR
~	• Bitwise NOT
<<	• Bitwise left shift
>>	• Bitwise right shift

```
>>> b = int('01001101', 2)
>>> a = int('00111100', 2)

>>> bin(a & b)
'0b1100'

>>> bin(a | b)
'0b1111101'

>>> bin(a ^ b)
'0b1110001'
```

Logical operators

and

or

Logical operators

```
>>> 0 or 500
500

>>> 0 or 500 or 1000
500

>>> [] and [1,2,3]
[]

>>> [1,2] and [1,2,3] and [1,2,3,4]
[1,2,3,4]

>>> None and True or (None, False) and [False]
```

Membership operators

in

not in

Membership operators

```
>>> 1 in [1, 2, 3]
True

>>> 4 in [1, 2, 3]
False

>>> 1 not in {'a', 'b', 'c'}
True

>>> 1 not in {1, 2, 3}
False
```

Identity operators

is

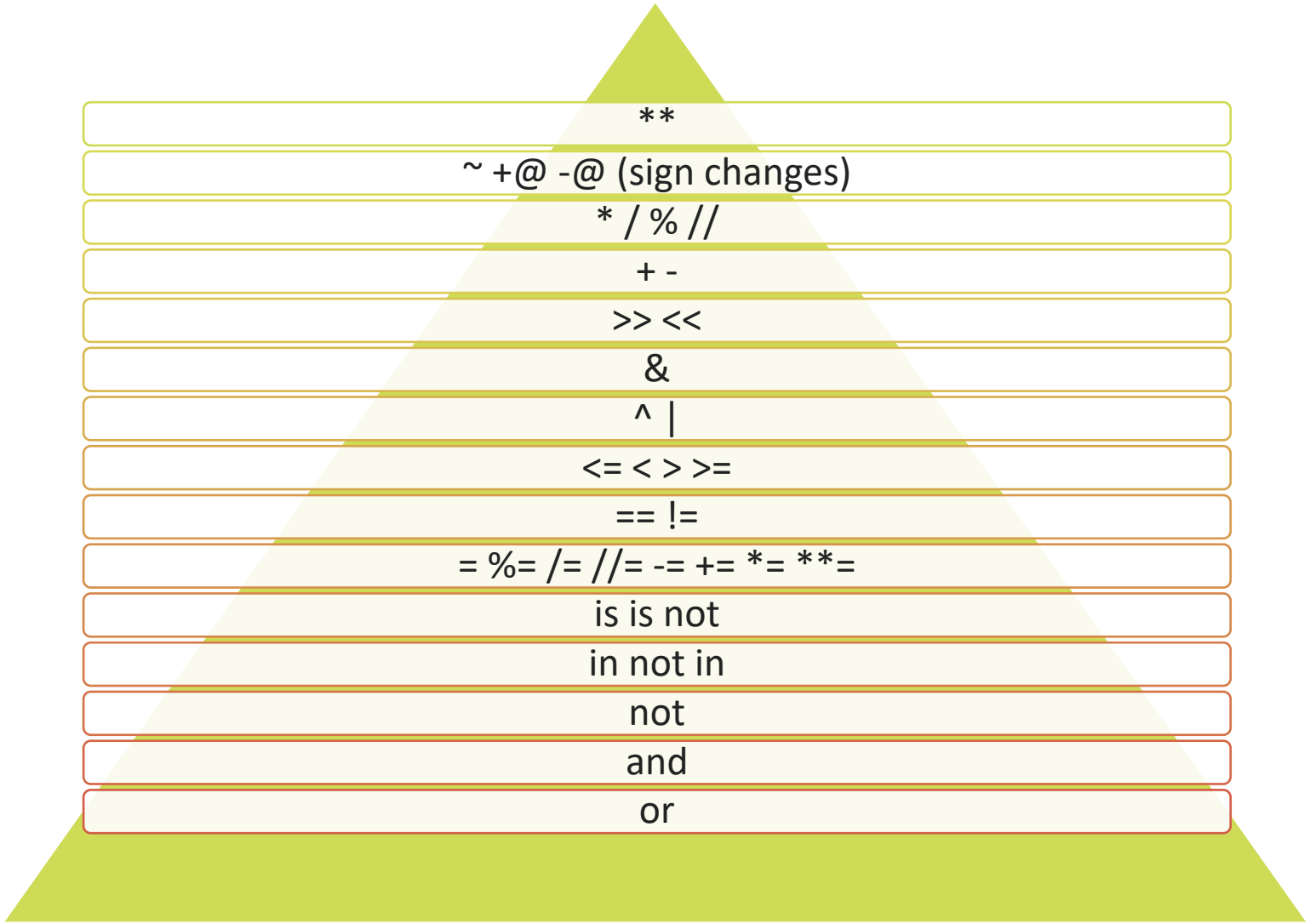
is not

Identity operators

```
>>> a = [1,2,3]
>>> b = a
>>> a.append(4)

>>> a is b
True

>>> print(a)
[1,2,3,4]

>>> print(b)
[1,2,3,4]
```

```
>>> a = "123"
>>> b = a
>>> a += "4"

>>> a is b
False

>>> print(a)
"1234"

>>> print(b)
"123"
```

Operator precedence



Conditions

Simple if statement

```
x = int(input('Input int: '))

if x < 0:
    print('x < 0')
```

If/else statement

```
x = int(input('Input int: '))

if x < 0:
    print('x < 0')
else:
    print('> 0')
```

If/elif/else statement

```
x = int(input('Input int: '))

if x < 0:
    print('x < 0')
elif x == 0:
    print('x equals 0')
else:
    print('> 0')
```

if statement

```
x = []

if x:
    print("True")
else:
    print("False")


x = [1]
if x:
    print("True")
else:
    print("False")
```

One-line if statement

Simple if statement

```
var = 100
if var == 100: print('Value of expression is 100')
print('Good bye!')
```

Ternary operator

```
>>> a = True
>>> b = 1 if a else 0
1
>>> a = False
>>> b = 1 if a else 0
0
```

Loops

Loop types

for

while

Loops usage

For loop syntax

```
for i in range(10):
    print(i)
```

While loop syntax

```
i = 10
while i > 0:
    print(i)
    i -= 1
```

Else block in loops

In `for` loops

```
for i in range(10):
    print(i)
else:  # end of collection
    print('End')
```

In `while` loops

```
while i > 0:
    print(i)
    i -= 1
else:  # i > 0 == False
    print('End')
```

Loops execution control

In `while` loops

```
i = int(input('int: '))

while i > 0:
    i -= 1
    if i == 5:
        print('5 - skipped')
        continue
    elif i == 1:
        print('1 - break')
        break  # no loop else
    else:
        print(i)
else:
    print('end')
```

Comprehensions

List comprehension

`for` loop

```
dogs= ['Gus', 'Bubba', 'Snoopy']

animals = []

for dog in dogs:
    animals.append(f'Dog {dog}')
```

List comprehension

```
dogs= ['Gus', 'Bubba', 'Snoopy']

animals = [
    f'Dog {dog}' for dog in dogs
]
```

Dict Comprehension

`for` loop

```
dog_owners = {
    'Adam': 'Gus',
    'Mike': 'Bubba',
    'Jessica': 'Snoopy',
}

animal_owners = {}

for owner, dog in dog_owners.items():
  animal_owners[owner] = f'Dog {dog}'
```

Dict comprehension

```
dog_owners = {
    'Adam': 'Gus',
    'Mike': 'Bubba',
    'Jessica': 'Snoopy',
}

animal_owners = {
    owner: f'Dog {dog}' for owner, dog
    in dog_owners.items()
}
```

Set comprehension

`for` loop

```
dogs= {'Gus', 'Bubba', 'Snoopy'}
cats = {'Bubba', 'Snow'}

unique_dog_names  = set()

for dog_name in dog_names:
   if dog_name not in cats:
        unique_dog_names.add(dog_name)
```

Set comprehension

```
dogs= {'Gus', 'Bubba', 'Snoopy'}
cats = {'Bubba', 'Snow'}

unique_dog_names = {
   dog_name for dog_name in dogs
   if dog_name not in cats
}
```

Algorithm complexity

Complexity of an algorithm is the amount of *time* or *space* required to run it. The factor of time is usually more important than that of space.

Time Complexity is most commonly estimated by **counting the number of elementary** *steps* performed to finish execution.

The big-O notation defines the **worst-case time complexity** of an algorithm.

Time complexity

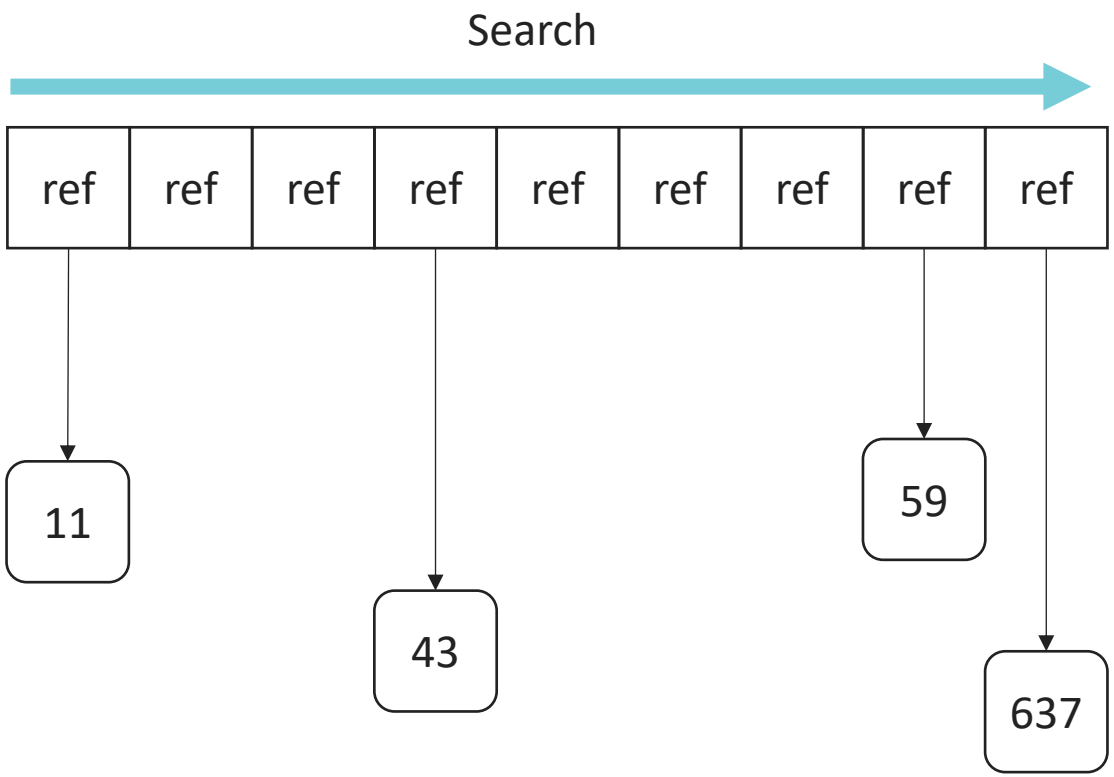
- $O(1)$ - constant
- $O(\log n)$ - logarithmic
- $O(n)$ - linear
- $O(n^2)$ - quadratic
- $O(2^n)$ - exponential
- etc.

Source: https://en.wikipedia.org/wiki/Time_complexity

Algorithm complexity example

```
# Find element in sorted array.
# Simple search.

array = [11,23,34,43,44,47,59,634]
element_to_find = 59

for item in array:
        if item == element_to_find:
                print("Found")
                break
else:
        print("Did not found")
```

Complexity – O(n)



Algorithm complexity example

dict	if k in d	$O(1)$
list	if x in l	$O(n)$
set	If x in s	$O(1)$

Source: https://wiki.python.org/moin/TimeComplexity

Thanks for attention

Questions?