ARTEZIO

Python: overview and data types

Short description

+ 1 212 220 1641
info@artezio.com
www.artezio.com

About Python

- Interpreted
- Procedural, object-oriented, functional
- All is object!

```
isinstance(5, object) # True
isinstance(print, object) # True
isinstance(object, object) # True
```

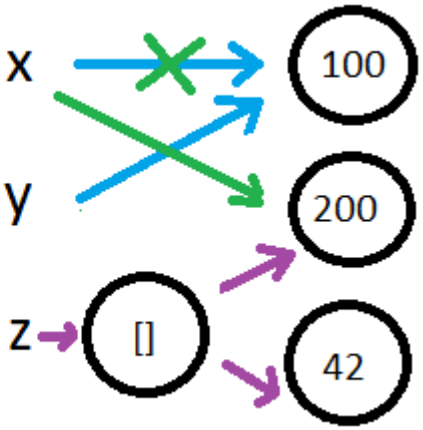
- Automatic memory handling

About Python

- Variables are just name bindings

```
x = 100 # 1
y = x # 2
x = 200 # 3
z = [x, 42] # 4
```

x = 100 (1)
y = x (2)
x = 200 (3)
z = [x, 42] (4)



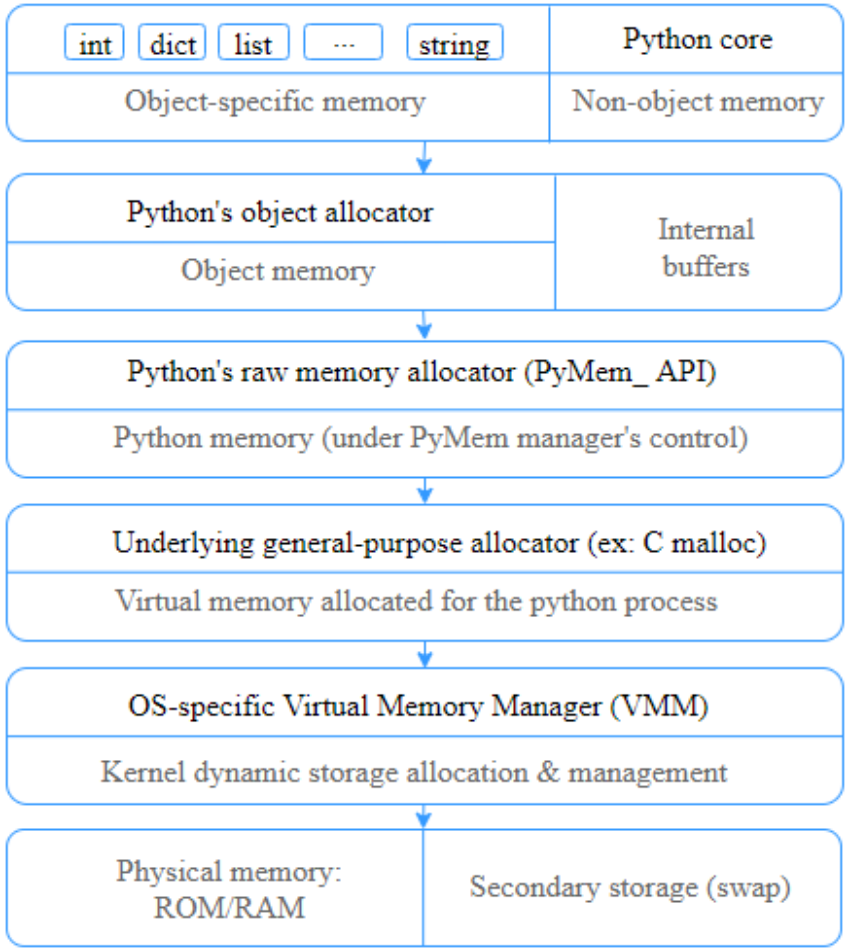
- Dynamically typed

```
a = 5
type(a) # <class 'int'>
a = 'hello!'
type(a) # <class 'str'>
```

- Awesome readability(pep-8, pep-20)

Memory model

- Python operates memory itself
- Different managers are used depending on the object size

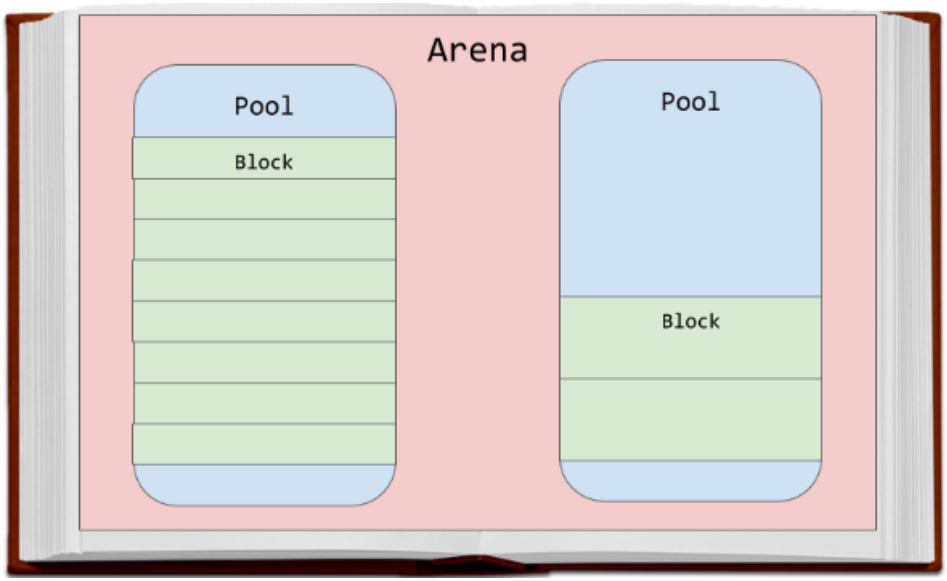


- Arena(256KB)
- Pool(4KB)
- Block(pool_type)

```
import sys
sys._debugmallocstats()
```

Creating new object

```
import sys
a = 'Helloworld'
sys.getrefcount(a) # 2
b = 'Helloworld'
sys.getrefcount(a) # 3
c = a
sys.getrefcount(a) # 4
```



Garbage collection

- Garbage collector

  Reference count

  ```
  a = 1 # 1 ref
  del a # 0 refs, clean up ASAP
  ```

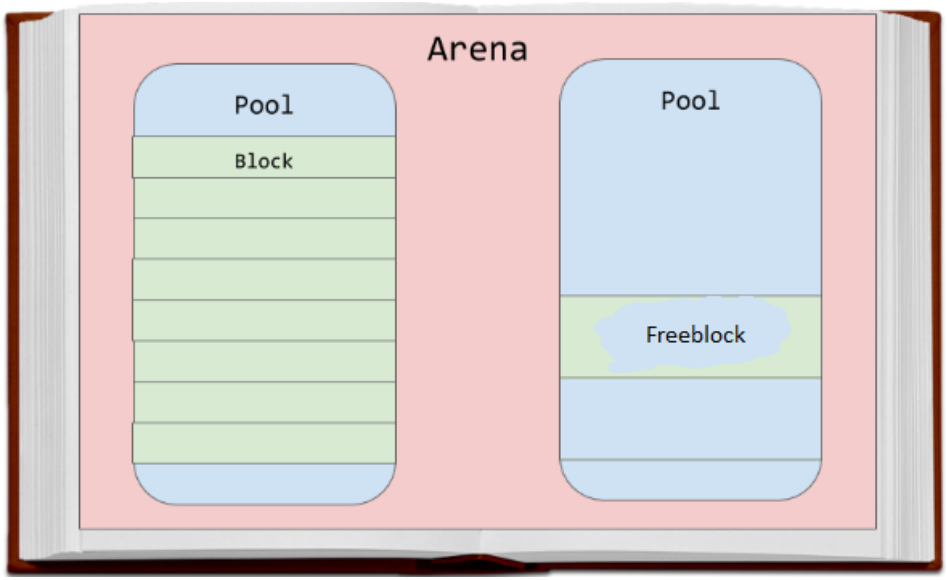
  Problem: cyclic references

  ```
  a = []
  b = []
  a.append(b)
  b.append(a) # 1
  ```

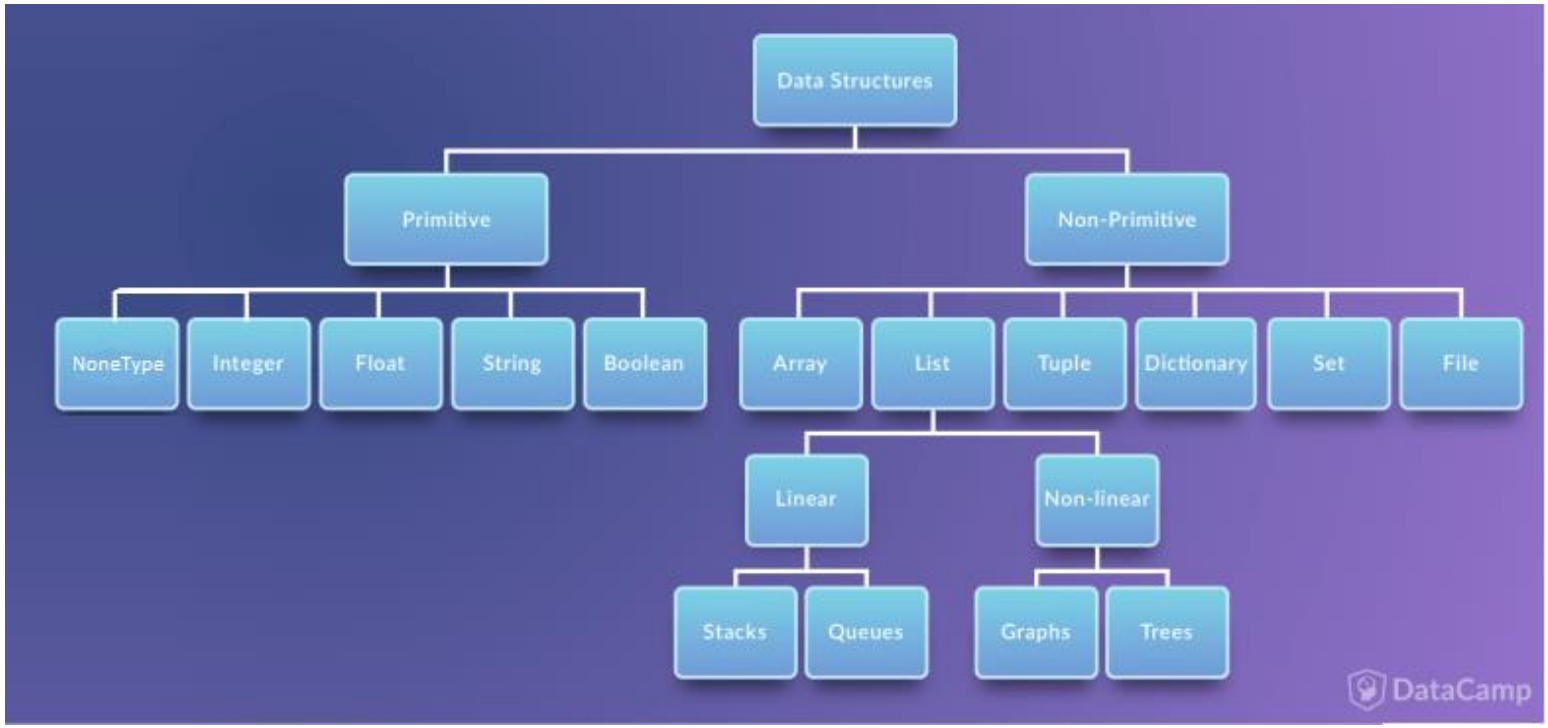
- Generational GC
  - Optional
  - Periodic execution
  - 3 generations of objects

Removing the object

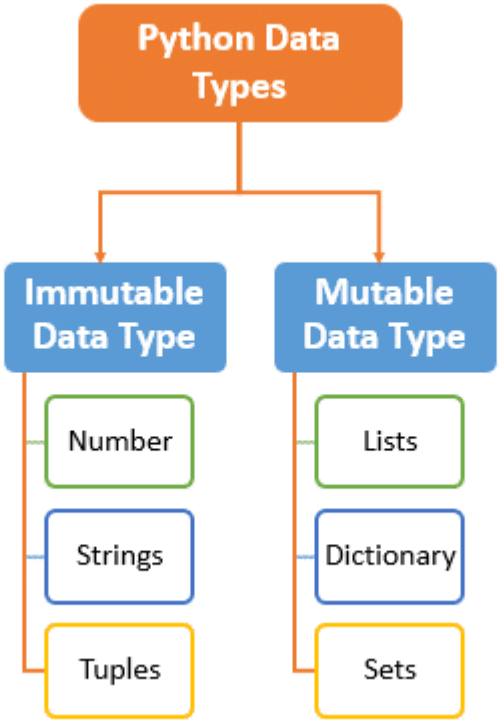
- If object is GC-ed -> block is added to **freeblock**
- If all blocks are free -> pool is added to **freepools**
- If all pools are free -> arena is free and returned to OS



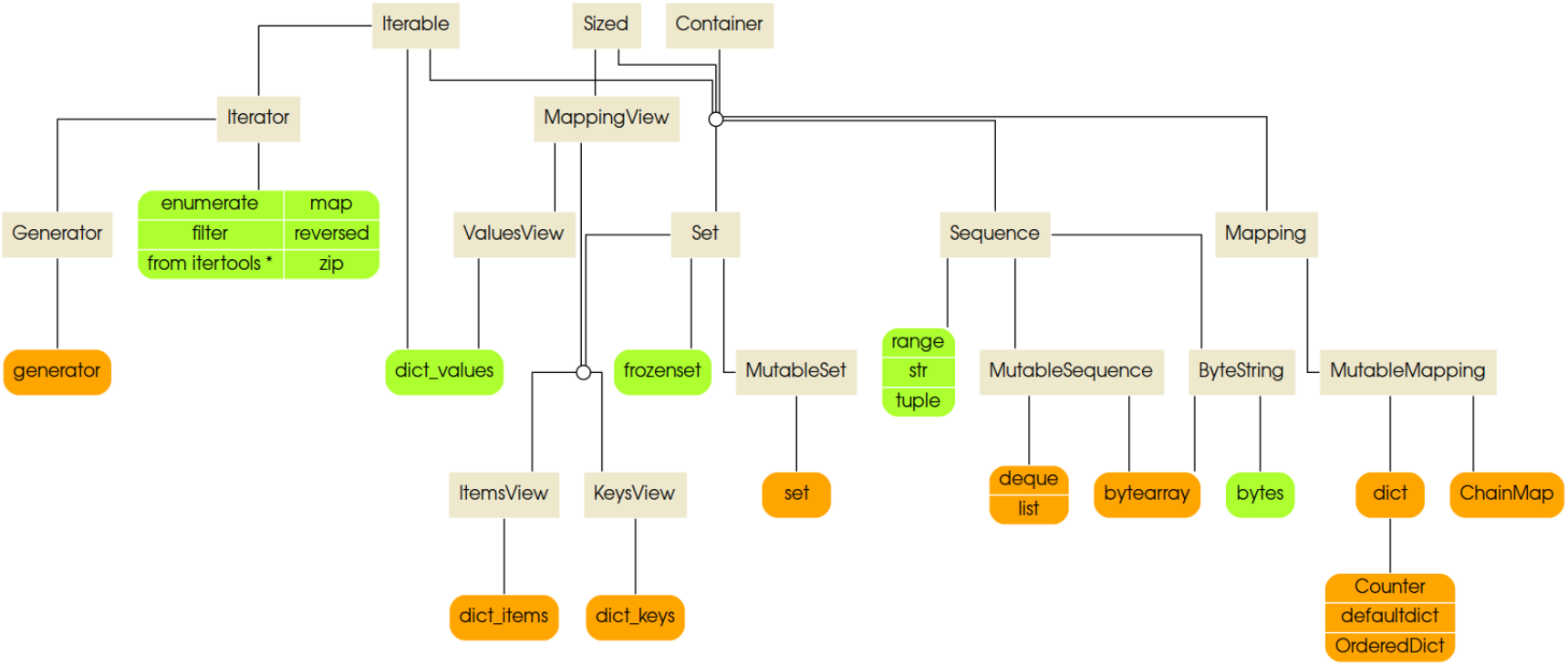
Data types


Data Structures
Primitive
Non-Primitive
NoneType
Integer
Float
String
Boolean
Array
List
Tuple
Dictionary
Set
File
Linear
Non-linear
Stacks
Queues
Graphs
Trees
DataCamp

Data types

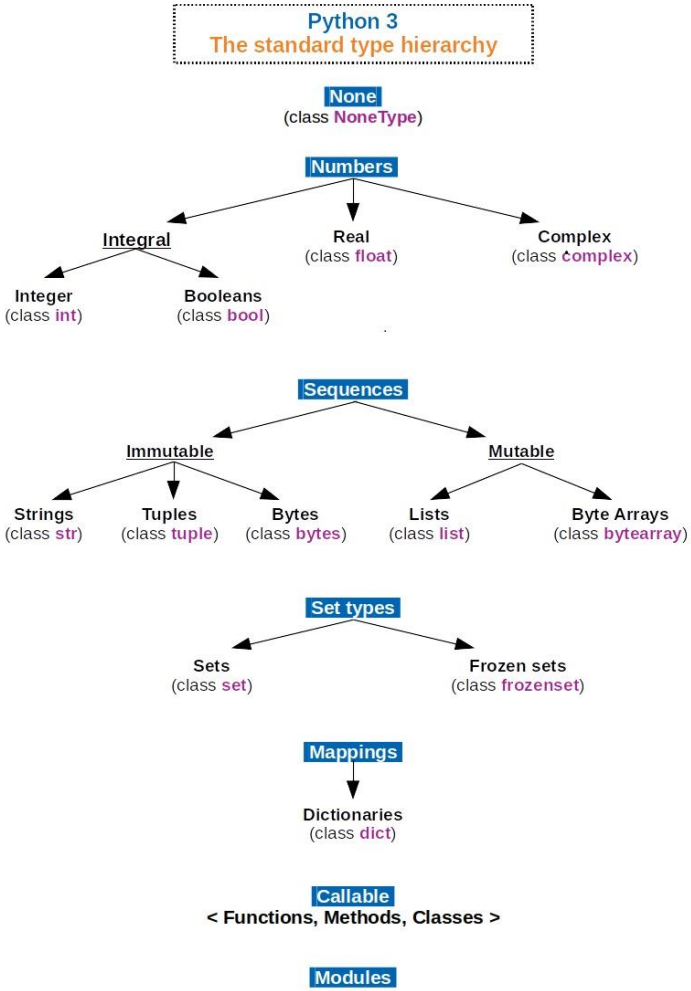

Python Data Types
Immutable Data Type
Mutable Data Type
Number
Strings
Tuples
Lists
Dictionary
Sets

Data types



Python 3 Collection Inheritance

Data types


Python 3
The standard type hierarchy
None
(class NoneType)
Numbers
Integral
Real
(class float)
Complex
(class complex)
Integer
(class int)
Booleans
(class bool)
Sequences
Immutable
Mutable
Strings
(class str)
Tuples
(class tuple)
Bytes
(class bytes)
Lists
(class list)
Byte Arrays
(class bytearray)
Set types
Sets
(class set)
Frozen sets
(class frozenset)
Mappings
Dictionaries
(class dict)
Callable
< Functions, Methods, Classes >
Modules

Object header

header

```
{
  object type(8 bytes)
  ref_count (8 bytes)
};
```

var_header

```
{
  object type(8 bytes)
  ref_count (8 bytes)
  size(8 bytes)
};
```

NoneType

```
val = None
```

```
{
  header(16 bytes)
};
```

Numbers: int, float, complex

```
val = 42
```

```
{
  var_header(24 bytes)
  payload(4 bytes)
};
```

```
val = 42.0
```

```
{
  header(16 bytes)
  payload(8 bytes)
};
```

Numbers: int, float, complex

General ops

- -x
- +x
- x + y
- x – y
- x * y
- x / y
- x // y
- x % y
- x ** y

Bitwise ops

- x | y
- x ^ y
- x & y
- x << n
- x >> n
- ~x

Additional methods

- abs(x)
- int(x)
- float(x)
- complex(re, im)
- c.conjugate()
- divmod(x, y)

bool

```
PyAPI_DATA(struct _longobject) _Py_FalseStruct, _Py_TrueStruct;
```

```
sys.getsizeof(False) == sys.getsizeof(0)
sys.getsizeof(True) == sys.getsizeof(1)
```

- **False, None, 0, “”, [], (), {}**
- **Any empty collections(set(),frozenset() etc)** → **False**
- **__bool__() returns False**
- **__len__() returns 0**

```
bool(()) # False
bool((),)) # True
```

str

```
mystring = 'hi world'
also_string = 'bye world'
yet_another_string = '''used for multi-lines and docs'''
```

- String literals: r, b, u, f, rf, rb
- Almost everything is casted to string(None, int, bool, object etc)
- (almost)Any string operations produce new string(don’t use + too much)
- Explicit type casting
- Slices

Common sequence methods

- x in s
- x not in s
- s1 + s2
- s * n or n * s
- s[i]
- s[i:j]
- s[i:j:k]
- len(s)
- min(s)
- max(s)
- s.index(x[, i[, j]])
- s.count(x)

bytes, bytearray

```
bytestring = b'hi world'
also_bytestring = bytes('bye world', encoding='utf8')
bytes.fromhex('686920776f726c64') # b'hi world'

ascii_bytearray = bytearray(b'Hi world!')
my_bytearray = bytearray('asd', encoding='utf8')
ascii_bytearray[-1] = ord('?')
ascii_bytearray # bytearray(b'Hi world?')
bytearray.fromhex('486921') # bytearray(b'Hi!')
```

Mutable sequence methods(list, bytearray)

```
s[i] = x
s[i:j] = t
del s[i:j]
s[i:j:k] = t
del s[i:j:k]
s.append(x)
s.clear()
```

```
s.copy()
s.extend(t) or s += t
s *= n
s.insert(i, x)
s.pop([i])
s.remove(x)
s.reverse()
```

Unicode

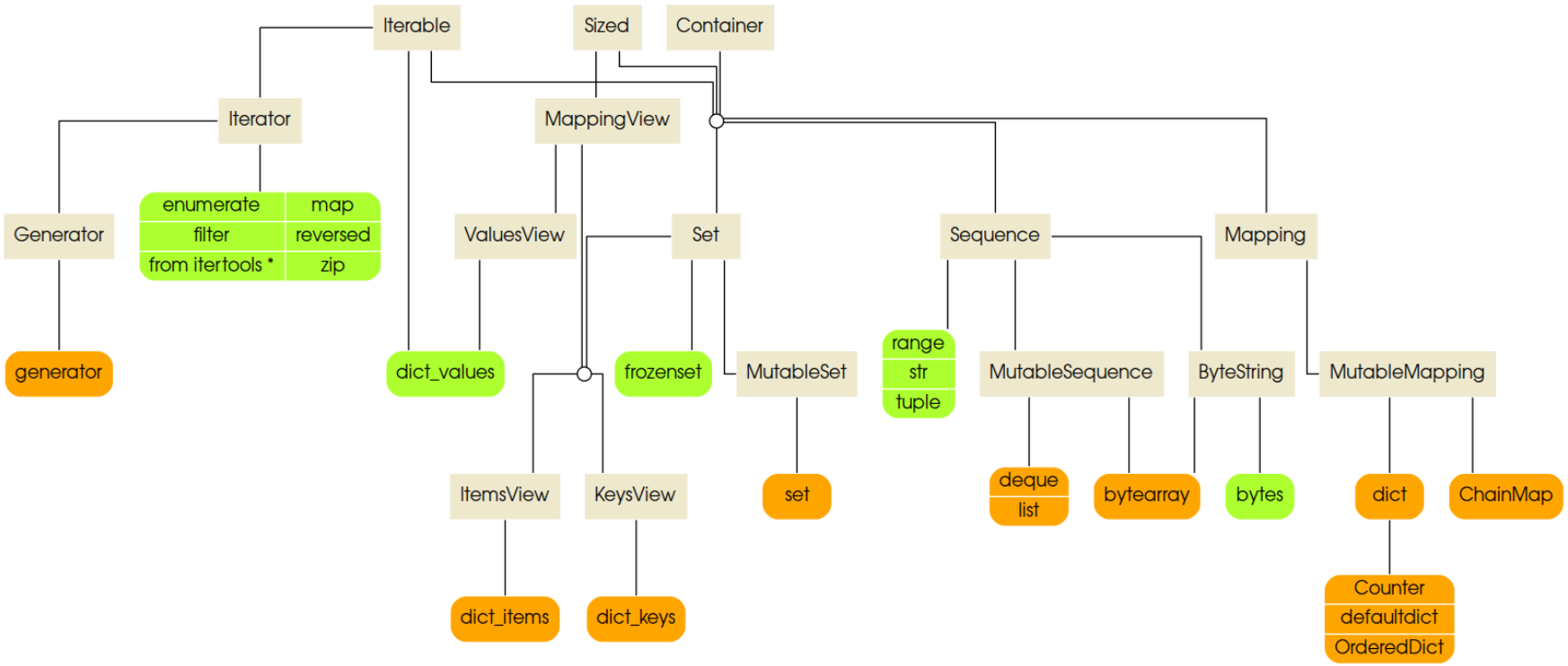
```
uc1 = 'hi world'
uc2 = 'hi \u265E'
uc4 = 'hi \U0001F609'
sys.getsizeof(uc1)  # 48 + len(uc1) * 1
sys.getsizeof(uc2) # 74 + len(uc2) * 2
sys.getsizeof(uc4) # 76 + len(uc4) * 4
```

- Strings in Python3 are Unicode…
  `'hello' == u'hello'` # True
- …Unless **b** literal is used
  ```
  b'hello' == u'hello' # False
  type(b'hello') # bytes
  ```
- String size depends on max Unicode point it contains
- Encodings – latin-1, utf8, utf16, utf32…

String methods

- capitalize
- casefold
- center
- count
- encode
- endswith
- expandtabs
- format
- format_map
- index
- isalnum
- isalpha
- isascii
- replace
- isdecimal
- isdigit
- isidentifier
- isnumeric
- isprintable
- isspace
- istitle
- isupper
- join
- ljust
- lower
- lip
- partition
- rfind
- rindex
- rjust
- rpartition
- rsplit
- rstrip
- split
- splitlines
- startswith
- strip
- swapcase
- title
- translate
- upper
- zfill

collections



Python 3 Collection Inheritance

…and some more

- collections
- function
- generator
- iterator
- class
- method
- module
- etc.

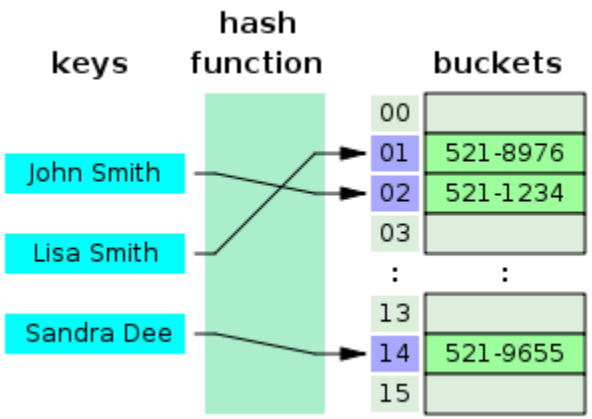
How to choose?

- Start size(already discussed)
- Big O notation
  - amount of CPU operations
  - amount of memory consumed

```
users_set = {1, 5, 132, 8, 54}
users_list = [1, 5, 8, 54, 132]
users_tuple = (1, 5, 8, 54, 132)

# CPU # find 5:
5 in users_set # O(1)
5 in users_list # O(n)
5 in users_tuple # O(n)
# memory
from pympler import asizeof
asizeof.asizeof(users_list) # 256
asizeof.asizeof(users_set) # 888
asizeof.asizeof(users_tuple) # 240
```

Hash table



	avg	worst
memory	O(n)	O(n)
find	O(1)	O(n)
insert	O(1)	O(n)
delete	O(1)	O(n)

- dict
- set
- frozenset
- …

Hashable objects

__hash__()
__eq__()

user-defined classes

```
class A:
    pass
a = A()
b = A()
hash(a) # f(id(a))
hash(b) # f(id(b))
a == b # False
a is b # False
```

Hashable objects

- numbers
```
hash(42) # 42
hash(42.0) # 42
hash(complex(42)) # 42
hash(2 ** 61 -1) # 0
```

- strings
```
a = "hello"
b = "hello"
hash(a) == hash(b) # True
```

- immutables: `hash((1,2,3))  # f(hash(1),hash(2),hash(3))`

- mutables
```
hash([1,2,3]) # error
hash({1,2,3}) # error
hash({1: 42}) # error
```

To sum up

1. Remember about memory consumption
2. Use right data types in right place
3. Be careful with mutables
4. Don’t confuse equality and identity
5. Learn by practice

References

1. https://www.python.org/dev/peps/pep-0020/
2. https://docs.python.org/3/reference/datamodel.html
3. https://docs.python.org/3/c-api/memory.html
4. https://github.com/python/.../obmalloc.c
5. https://habr.com/ru/post/417215/
6. https://github.com/python/.../gcmodule.c
7. https://www.youtube.com/memory_in_python
8. https://nedbatchelder.com/text/names.html
9. https://docs.python.org/3/howto/unicode.html
10. https://rushter.com/blog/python-strings-and-memory/
11. https://docs.python.org/3/library/collections.abc.html
12. https://docs.python.org/3/library/stdtypes.html
13. https://www.python.org/dev/peps/pep-0008/
14. https://wiki.python.org/moin/TimeComplexity
15. https://en.wikipedia.org/wiki/Hash_table
16. https://github.com/python/.../pyhash.c
17. https://fengsp.github.io/blog/2017/3/python-dictionary/

ARTEZIO

Thanks for your attention

artezio_software
info@artezio.com

www.artezio.com