Data structures

Konov Ilya
phd, computer engineering

girafe ai

Outline

Lecture 2:
Data structures

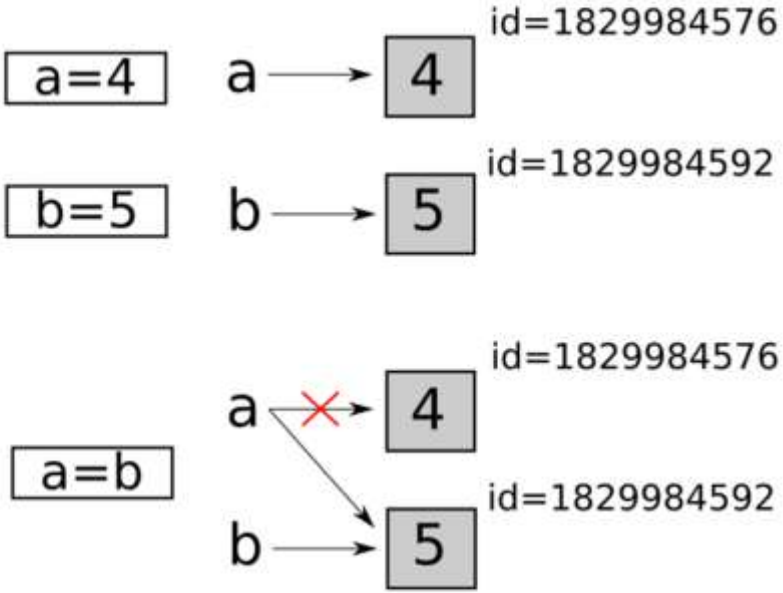
- Basic data structures
  - int
  - float
  - complex
- Compound data types
  - list
  - tupl
  - set
- Basic control Flow Tools

Basic data structures

girafe
ai

01

Variables and references



Basic Data structures

- Numeric Type
    - int – integral number
    - float – floating point number
    - complex – complex number
- String (Text Sequence Type)
- None (undefined value of the variable)
- Bool-(boolean type)
- Sequence Type
    - list
    - tuple
    - range
- Set Types
    - set
    - frozenset
- Mapping Types
    - dict – dictionary

int

Integer number

min val =-9223372036854775808

max val = 9223372036854775807

Basic operations

(+,-,*,/)

Additional Methods on Integer Types

int.bit_length()

int.to_bytes()

int.from_bytes()

Bitwise operation

Operation	Description
x | y	bitwise OR
x ^ y	logical exclusive OR
X & y	bitwise AND
x << n	bits shifted to the left by y places
x >> n	bits shifted to the right by n places
~x	switching each 1 for a 0 and each 0 for a 1

float

floating point number

(+-*/)

Floating-point numbers are represented in computer hardware as base 2 (binary) fractions.

For example

0.3 ~ 1/4 + 1/16 ...

```
0.1 + 0.1 + 0.1 == 0.3
✓ 0.0s

False
```

https://docs.python.org/3/tutorial/floatingpoint.html

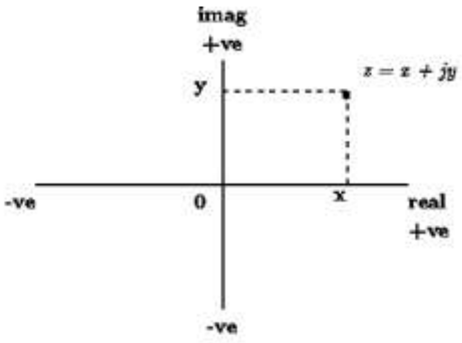
Complex number

```c
typedef struct {
    double real;
    double imag;
} Py_complex;
```

```python
complex_number = 1 + 2j
print(complex_number.real)
print(complex_number.imag)
```

✓ 0.0s

```
1.0
2.0
```



Basic operations

(+,-,*,/)

Additional Methods

cmath()

cmath.exp(x)

cmath.sin(x)

string

Textual data in Python is handled with str objects, or strings. Strings are immutable sequences of **Unicode** code points.

Single quotes: 'allows embedded "double" quotes'

Double quotes: "allows embedded 'single' quotes"

Triple quoted: '''Three single quotes''', """Three double quotes"""

```
string_value = 'Text'
print(string_value[0])
print(string_value[0:1])
print(string_value[0:3])
```

✓ 0.0s

```
T
T
Tex
```

String Methods

str.capitalize().

str.islower()

str.replace.

Slice

str[star:end:step].

```
string_value = 'Text'
print(string_value[0])
print(string_value[0:1])
print(string_value[0:3])
✓ 0.0s

T
T
Tex
```

format и f’string’

```
day = 9
manual_string = 'Today is the '
print(manual_string,day) #Today is the  9
print(manual_string + str(a))  #Today is the 9
print(f'Today is the {day}')  #Today is the 9
print('Today is the {}'.format(day)) #Today is the 9
```

Format Specification Mini-Language

```
>>> '{:<30}'.format('left aligned')
'left aligned                  '
>>> '{:>30}'.format('right aligned')
'                 right aligned'
>>> '{:^30}'.format('centered')
'           centered           '
>>> '{:*^30}'.format('centered')  # use '*' as a fill char
'***********centered***********'
```

```
>>> points = 19
>>> total = 22
>>> 'Correct answers: {:.2%}'.format(points/total)
'Correct answers: 86.36%'
```

https://docs.python.org/3/library/string.html#formatspec

Bool

True or False

True == 1

False == 0

Support Boolean algebra

Logical operation	Operator	Notation	Alternative notations	Definition
Conjunction	AND	$x \wedge y$	x AND y, Kxy	$x \wedge y = 1$ if $x = y = 1$, $x \wedge y = 0$ otherwise
Disjunction	OR	$x \vee y$	x OR y, Axy	$x \vee y = 0$ if $x = y = 0$, $x \vee y = 1$ otherwise
Negation	NOT	$\neg x$	NOT x, Nx, $\bar{x}$, x', $!x$	$\neg x = 0$ if $x = 1$, $\neg x = 1$ if $x = 0$

https://en.wikipedia.org/wiki/Boolean_algebra

None

The None object is an object that is used to represent the absence of a value in Python.

- Show the absence of a value when a variable does not have a specific value
- To indicate that the function does not return any value
- Use as a placeholder when creating lists, dictionaries, and other data structures

```
a = 1
def function1(a):
    a = a + 1
b = function1(a)
print(b is None)
```

✓ 0.0s

```
True
```

Compound data types

girafe ai

02

List

Lists are mutable sequences, typically used to store collections of homogeneous items

```
list_el = []
list_el = [1,2,3]
list_el = [1,'el',3]
```

List

Operation	Result
s[i] = x	item i of s is replaced by x
s[i:j] = t	slice of s from i to j is replaced by the contents of the iterable t
del s[i:j]	same as s[i:j] = []
s[i:j:k] = t	the elements of s[i:j:k] are replaced by those of t
s.append(x)	appends x to the end of the sequence
s.clear()	removes all items from s
s.insert(i, x)	inserts x into s at the index given by i (same as s[i:i] = [x])

Most of the list methods

```
>>> fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
>>> fruits.count('apple')
2
>>> fruits.count('tangerine')
0
>>> fruits.index('banana')
3
>>> fruits.index('banana', 4)  # Find next banana starting at position 4
6
>>> fruits.reverse()
>>> fruits
['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', 'orange']
>>> fruits.append('grape')
>>> fruits
['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', 'orange', 'grape']
>>> fruits.sort()
>>> fruits
['apple', 'apple', 'banana', 'banana', 'grape', 'kiwi', 'orange', 'pear']
>>> fruits.pop()
'pear'
```

Tupl

Tuples are immutable sequences, typically used to store collections of heterogeneous data

```
tupl_el = ()
tupl_el = (1,2,3)
tupl_el = (1,'el',3)
```

Common Sequence Operations

Operation	Result
x in s	True if an item of s is equal to x, else False
x not in s	slice of s from i to j is replaced by the contents of the iterable t
s[i]	item of s, from i
s[i:j]	slice of s from i to j
s[i:j:k]	slice of s from i to j with step k
len(s)	length of s
min(s)	smallest item of s
s.count(x)	total number of occurrences of x in s

What is the difference between list and tupl?

girafe ai

mutable	immutable
object can be changed after creation	object cannot be changed after creation
Access to mutable objects is slower compared to immutable objects	Access to immutable objects is faster compared to mutable objects
It is better when you need to change the size or content	Immutable objects are best suited when we are sure that we do not need to change them at any point in time.
Changing mutable objects is faster than changing non-mutable ones	To make a change, you need to create a new immutable object and make a change

Ranges

The range type represents an immutable sequence of numbers and is commonly used for looping a specific number of times in for loops.

class range(stop)

class range(start, stop[, step])

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(1, 11))
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list(range(0, 30, 5))
[0, 5, 10, 15, 20, 25]
>>> list(range(0, 10, 3))
[0, 3, 6, 9]
>>> list(range(0, -10, -1))
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
>>> list(range(0))
[]
>>> list(range(1, 0))
[]
```

Set

A set is an **unordered** collection with no duplicate elements. Basic uses include membership testing and eliminating duplicate entries.

Set objects also support mathematical operations like union, intersection, difference, and symmetric difference.

```
>>> basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
>>> print(basket)                      # show that duplicates have been removed
{'orange', 'banana', 'pear', 'apple'}
>>> 'orange' in basket                 # fast membership testing
True
>>> 'crabgrass' in basket
False

>>> # Demonstrate set operations on unique letters from two words
>>>
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a                                  # unique letters in a
{'a', 'r', 'b', 'c', 'd'}
>>> a - b                              # letters in a but not in b
{'r', 'd', 'b'}
>>> a | b                              # letters in a or b or both
{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}
>>> a & b                              # letters in both a and b
{'a', 'c'}
>>> a ^ b                              # letters in a or b but not both
{'r', 'd', 'b', 'm', 'z', 'l'}
```

dict

A dict is a mutable collection. Dictionary is a set of key: value pairs, with the requirement that the keys are unique (within one dictionary).

```
>>> a = dict(one=1, two=2, three=3)
>>> b = {'one': 1, 'two': 2, 'three': 3}
>>> c = dict(zip(['one', 'two', 'three'], [1, 2, 3]))
>>> d = dict([('two', 2), ('one', 1), ('three', 3)])
>>> e = dict({'three': 3, 'one': 1, 'two': 2})
>>> f = dict({'one': 1, 'three': 3}, two=2)
>>> a == b == c == d == e == f
True
```

dict

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'jack': 4098, 'sape': 4139, 'guido': 4127}
>>> tel['jack']
4098
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'jack': 4098, 'guido': 4127, 'irv': 4127}
>>> list(tel)
['jack', 'guido', 'irv']
>>> sorted(tel)
['guido', 'irv', 'jack']
>>> 'guido' in tel
True
>>> 'jack' not in tel
False
```

Dictionary view objects

```
>>> dishes = {'eggs': 2, 'sausage': 1, 'bacon': 1, 'spam': 500}
>>> keys = dishes.keys()
>>> values = dishes.values()

>>> # iteration
>>> n = 0
>>> for val in values:
...     n += val
...
>>> print(n)
504

>>> # keys and values are iterated over in the same order (insertion order)
>>> list(keys)
['eggs', 'sausage', 'bacon', 'spam']
>>> list(values)
[2, 1, 1, 500]

>>> # view objects are dynamic and reflect dict changes
>>> del dishes['eggs']
>>> del dishes['sausage']
>>> list(keys)
['bacon', 'spam']
```

Link Features

- Python doesn't copy anything until we ask it to - everything is passed "by reference"

```python
import copy

a = [1]
c = a

c.append(2)

print(a) #return [1,2]

#Copy obj
b = copy.copy(a)
#or
b = copy.deepcopy(a)
b = a[:]

b.append(2)
print(a) #return [1,2]
```

Example of a link counter

```
import sys

class TexiOrder:
    def __init__(self,name):
        self.name = name

    def __del__(self):
        print(f'Order {self.name} will be delete')

a = TexiOrder('some_order')


print(sys.getrefcount(a))

b = a

print(sys.getrefcount(a))

del b

print(sys.getrefcount(a))

del a
```

Example of a link counter

```python
import sys

class TexiOrder:
    def __init__(self,name):
        self.name = name

    def __del__(self):
        print(f'Order {self.name} will be delete')

a = TexiOrder('some_order')


print(sys.getrefcount(a)) # return 2

b = a

print(sys.getrefcount(a)) # return 3

del b

print(sys.getrefcount(a)) # return 2

del a # Order some_order will be delete
```

Basic control Flow Tools

girafe ai

03

if

```
if condition:
	# Code to execute if condition is True
```

Branching Operators if/else

```
if condition:
	# Code to execute if condition is True
else:
		# Code to execute if condition is False
```

Nested Conditional Statements

```
if condition_1:
    # Code to execute if condition_1 is True
    if condition_2:
        #Code to execute if condition_1 and  condition_2 are True
    …
        if condition_n:
                #Code to execute if all condition_1 and  condition_n are True
else:
        # Code to execute if condition is False
```

Nested Conditional Statements

```
if condition_1:
        # Code to execute if condition_1 is True
elif condition_2:
                #Code to execute  condition_2 is True
…

elif condition_n:
                                #ode to execute  condition_n is True

else:
                # Code to execute if condition is False
```

Common Sequence Operations

Operation	Result
list(d)	Return a list of all the keys used in the dictionary d.
len(d)	Return the number of items in the dictionary d.
d[key]	Return the item of d with key key.
d[key] = value	Set d[key] to value.
del d[key]	Remove d[key] from d.
key in d\key not in d	Return True if d has a key key, else False.
iter(d)	Return an iterator over the keys of the dictionary.
clear()	Remove all items from the dictionary.

Thanks for attention!

Questions?

girafe
ai

PyTypeObject

- Base class for all types
  - Basically, it contains pointers to C functions, (__hash__, __str__, __new__, __init__)

PyTypeObject {
PyObject_VAR_HEAD
const char *tp_name; /* For printing, in fo
rmat "<module>.<name>" */
/* ... */
PyNumberMethods *tp_as_number;
/* ... */
hashfunc tp_hash;
ternaryfunc tp_call;
reprfunc tp_str;
initproc tp_init;
newfunc tp_new;
/* ... */
} PyTypeObject

PyObject

- PyObject - the main type of all objects

```
typedef struct _object {
        /* ... */
         Py_ssize_t ob_refcnt; struct
_typeobject *ob_type; } PyObject
```

Other objects are "inherited" from it, for example -

float

```
typedef struct {
 PyObject_HEAD /* a structure
above */
double ob_fval;
} PyFloatObject;
```