Python for Data Scientists
L2: Data types and Data Structure in Python

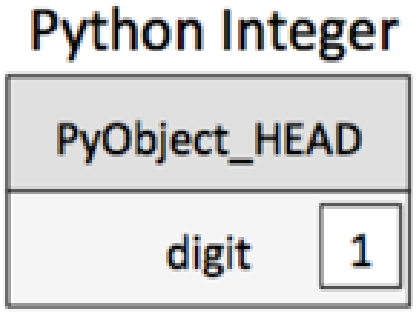
Variables

- When creating a variable, the interpreter will reserve some space in the memory to store values.

- Python variables are usually dynamically typed

```python
# python code
a = 1
b = 2
c = a + b
```

```c
/* C code */
int a = 1;
int b = 2;
int c = a + b;
```



Variables

There are certain rules or naming conventions for naming variables:

- Reserved key words such as if, else, and so on cannot be used for naming variables
- Variable names can begin with _, $, or a letter
- Variable names can be in lower case and uppercase
- Variable names cannot start with a number
- White space characters are not allowed in the naming of a variable

Variables

Syntax:

<variable name>= < expression >

Single assignment

- city='London’
- money = 100.75
- count=4

Multiple assignment

- a = b = c = 1

Data types

- Refers to a given type along with a collection of operations for manipulating values of the given type.

- Programming languages commonly provide data types as part of the language itself.

Data types

Python has the following data types built-in by default, in these categories:

- Text type: str
- Numeric types: int, float
- Sequence types: list, tuple
- Mapping type: dict
- Set Type: set
- Boolean type: bool

Data Structure

- A particular way of storing and organizing data in a computer so that it can be used efficiently and effectively.

- A group of data elements grouped together under one name (example: an array of integers)

Numeric: Integers

Integers : Zero, positive and negative whole numbers without a fractional part

- ranges: -2^{31} to (2^{31}-1)) and long integers

Numeric: float

Float: Positive and negative real numbers with a fractional part denoted by the decimal symbol or the scientific notation using e

- ranges approximately : -10 to 10^{308} and has 16 digits of precision.
- Example : 5,32 or 532e-2

Numeric

Arithmetic Operators

- + : Addition
- - : Subtraction
- * : Multiplication
- / : Division
- % : Modulus
- ** : Exponent
- // : Floor division

Boolean

Boolean data type

- <variable name>=<'True' or 'False’>

Boolean

Logic operators on Booleans values

- not x -> True if x is False or False if x is True
- x and y -> True if both are True
- x or y -> True if either or both are true

String

String data types

- <variable name>= <String sequence>
- Single quotes ‘ ’ or double quotes “ ” can be used to denote a string

Example:

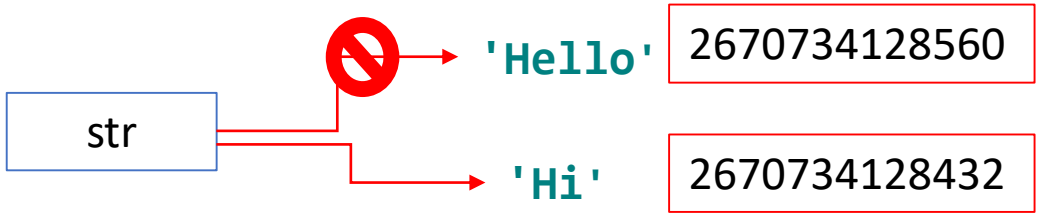
```
print('Welcome to the "Python for Data Scientists"
course')
print("Welcome to the 'Python for Data Scientists'
course")
```

Welcome to the "Python for Data Scientists" course
Welcome to the 'Python for Data Scientists' course

Strings

String is an immutable:

```
str = 'Hello'
print(id(str))
str = 'Hi'
print(id(str))
```



Strings

Square brackets used to perform **indexing** into a string to get the value at a certain position (index).

```
film = 'Joker’

Index:  01234  -> indexing starts at 0

Index: -5-4-3-2-1   -> last element at index -1

film[0]   -> evaluates to ‘J’

film[-1]  -> evaluates to ‘r’
```

→ Similar to other programming languages, strings in Python are arrays of bytes representing unicode characters.
→ Python does not have a character data type (a single character is a string with a length of 1).

Strings

Slicing for substrings

- can **slice** strings using [start:stop:step]
- if give two numbers, [start:stop], step=1 by default

```
film = 'Joker'
print(film[0:2])      -> Jo
print(film[:3])       -> Jok
print(film[3:])       -> er
print(film[::3])      -> Je
print(film[::])       -> Joker
print(film[::-2])     -> rkJ
```

Strings (Poll)

Example: suppose we have the following string : film=‘Joker’ and we would like to replace the first upper case letter with a lower case.

Which of the flowing statement is correct

```
A- film[0]='j’    -> TypeError: 'str' object does not support item assignment
```

```
B- film='j'+film[1:]
```

Strings

String methods:

```
film = 'joker jokeR'
print(film.count('o'))
print(film.count('o',3,5))
print(film.find("ke"))
print(film.rfind("ke"))
print(film.lower())
print(film.upper())
print(film.capitalize())
print(film.title())
print(film.swapcase())
```

```
2
0
2
8
joker joker
JOKER JOKER
Joker joker
Joker Joker
JOKER JOKEr
```

All string methods do not change the original string (returns new values)

Conversion functions

- ord() : convert a character value to ASCII code
- chr() : convert ASCII code to character
- int() : convert into the int data type
- float() : convert into the float data type

Arrays

- Arrays in Python are a compact way of collecting basic data types (all the entries in an array must be of the same data type).

- Arrays are not all that popular in Python, unlike the other programming languages such as C++ or Java.

- For python, you need to import a module named “**array**“

Arrays

- **array(data type, value list)** : create an array with data type and value list specified in its arguments.

- **append()** : add the value mentioned in its arguments at the end of the array.

- **insert(i,x)** : add the value at the position specified in its argument.

Arrays

- **pop()** : removes the element at the position mentioned in its argument and returns it.

- **remove()** : remove the first occurrence of the value mentioned in its arguments.

- **index()** : returns the index of the first occurrence of value mentioned in arguments.

- **reverse()** : reverses the array.

Arrays

```
import array
arr = array.array('i', [1, 5, 8])

print("The new created array is : ", end=" ")
for i in range(0, 3):
    print(arr[i], end=" ")

print("\r")
arr.append(0);
print("The appended array is : ", end="")
for i in range(0, 4):
    print(arr[i], end=" ")

print("\r")
arr.insert(3, 7)
print("The array after insertion is : ", end="")
for i in range(0, 5):
    print(arr[i], end=" ")
```

```
The new created array is :  1 5 8
The appended array is : 1 5 8 0
The array after insertion is : 1 5 8 7 0
```

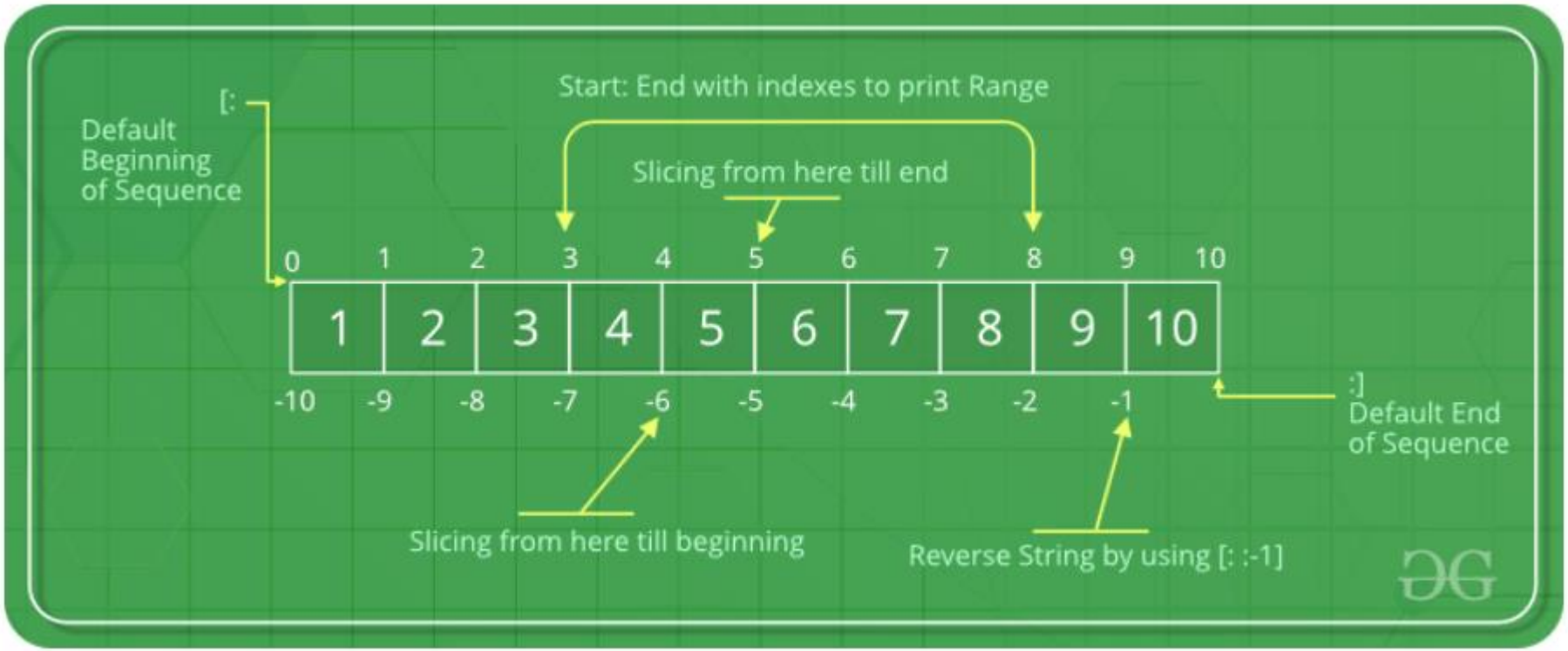
Arrays

```
print("The popped element is : ", end="")
print(arr.pop(2));
print ("The array after popping is : ",end="")
for i in range (0,4):
    print (arr[i],end=" ")

print("\r")
arr.remove(1)
arr.reverse()
print("The array after removing is : ", end="")
for i in range(0, 3):
    print(arr[i], end=" ")
```

The popped element is : 8
The array after popping is : 1 5 7 0
The array after removing is : 0 7 5

Arrays: Slicing



https://www.geeksforgeeks.org/python-arrays/

Numpy arrays

- provide a high-performance multidimensional array object, and tools for working with these arrays.
- NumPy's library of algorithms written in C can operate on this memory without any type checking or other overhead
- use much less memory than built-in Python sequences
- NumPy operations perform complex computations on entire arrays without the need for python loops

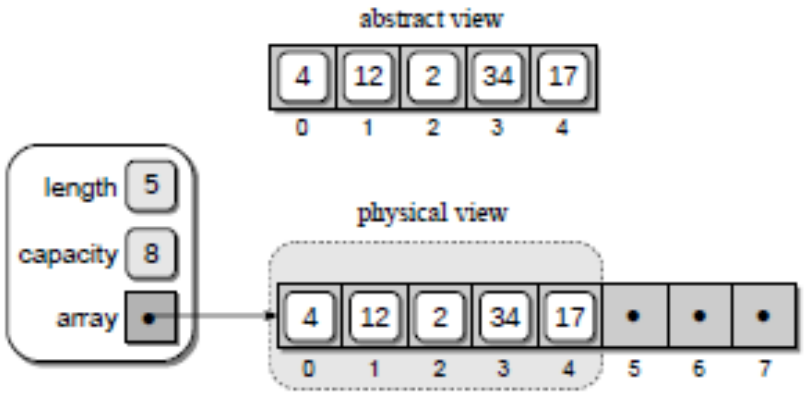
Lists

- ordered sequence of information, accessible by index
- a list is denoted by square brackets, []
- a list contains elements
  - usually homogeneous (ie, all integers)
  - can contain mixed types (not common)

```
[1,2,3,4]
['Joker', 'Titanic', 'Ace Age']
[2.5, 'Joker', [1,2,3]]
```

Lists

In Python, lists are similar to arrays:



The abstract and physical views of a list implemented using an array

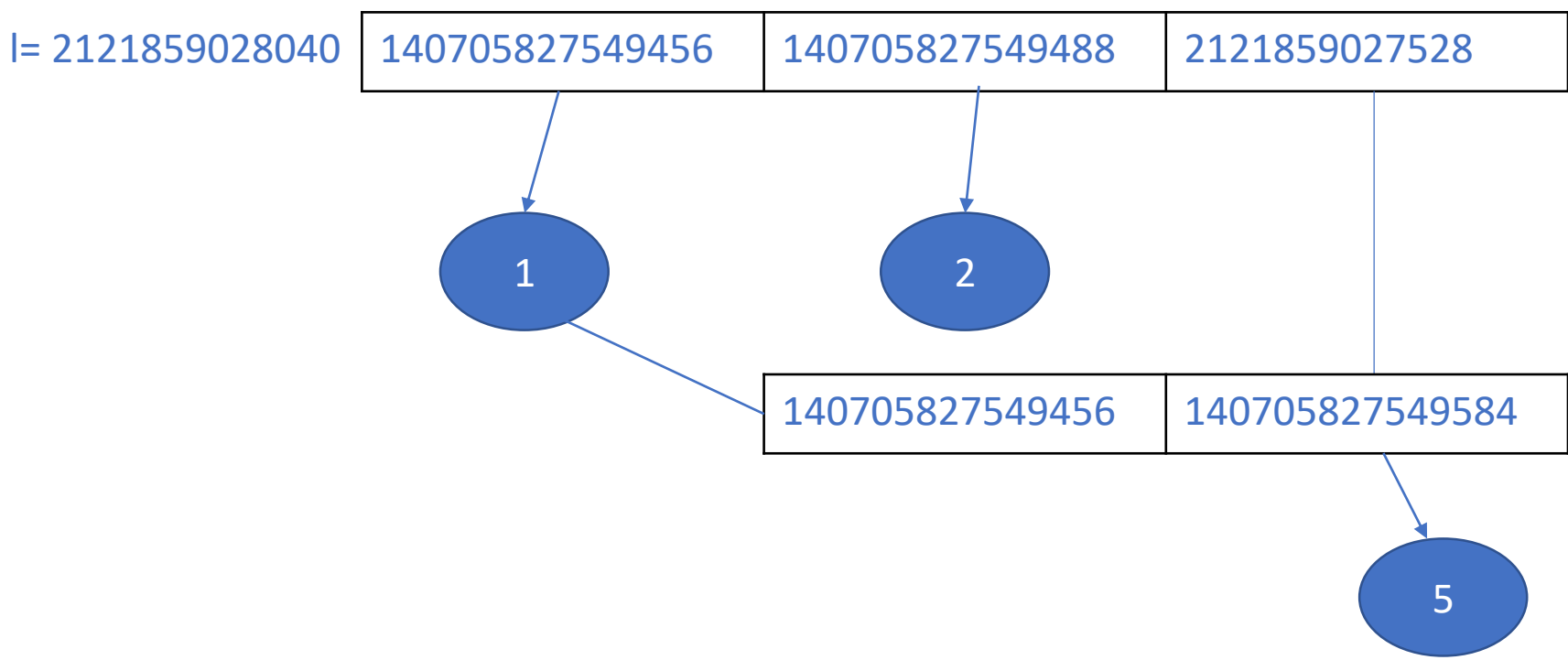
Book: Data structure and Algorithms using Python, Rance D.Necaise, ISBN 978-0-470-61829-5.

Lists

Memory allocation of Elements in List:

```
l = [1,2,[1,5]]
print(id(l[0]))
print(id(l[1]))
print(id(l[2]))
print(id(l[2][0]))
print(id(l[2][1]))
```

```
2121859028040
140705827549456
140705827549488
2121859027528
140705827549456
140705827549584
```



Lists

- Lists are mutable

```
numbers = [1,2,3,4]
numbers[0]=5
print(numbers)
```

[5, 2, 3, 4]

- in operator

```
print(5 in numbers)
```

True

Lists

- List operations

```
a = [1, 2, 3]
b = [4, 5, 6]
c = a + b
d = a * 3
```

c=[1, 2, 3, 4, 5, 6]
d=[1, 2, 3, 1, 2, 3, 1, 2, 3]

- List slices

```
c[1:3]
c[:3]
c[3:]
c[:]
```

[2, 3]
[1, 2, 3]
[4, 5, 6]
[1, 2, 3, 4, 5, 6]

Lists

List methods

- append: adds a new element to the end of a list
- extend: takes a list as an argument and appends all of the elements
- sort: arranges the elements of the list from low to high

```
l = ['a', 'b', 'c']
l.append('d’)

l2 = ['e', 'd']
l.extend(l2)
```

['a', 'b', 'c', 'd']
['a', 'b', 'c', 'd', 'e', 'd'']

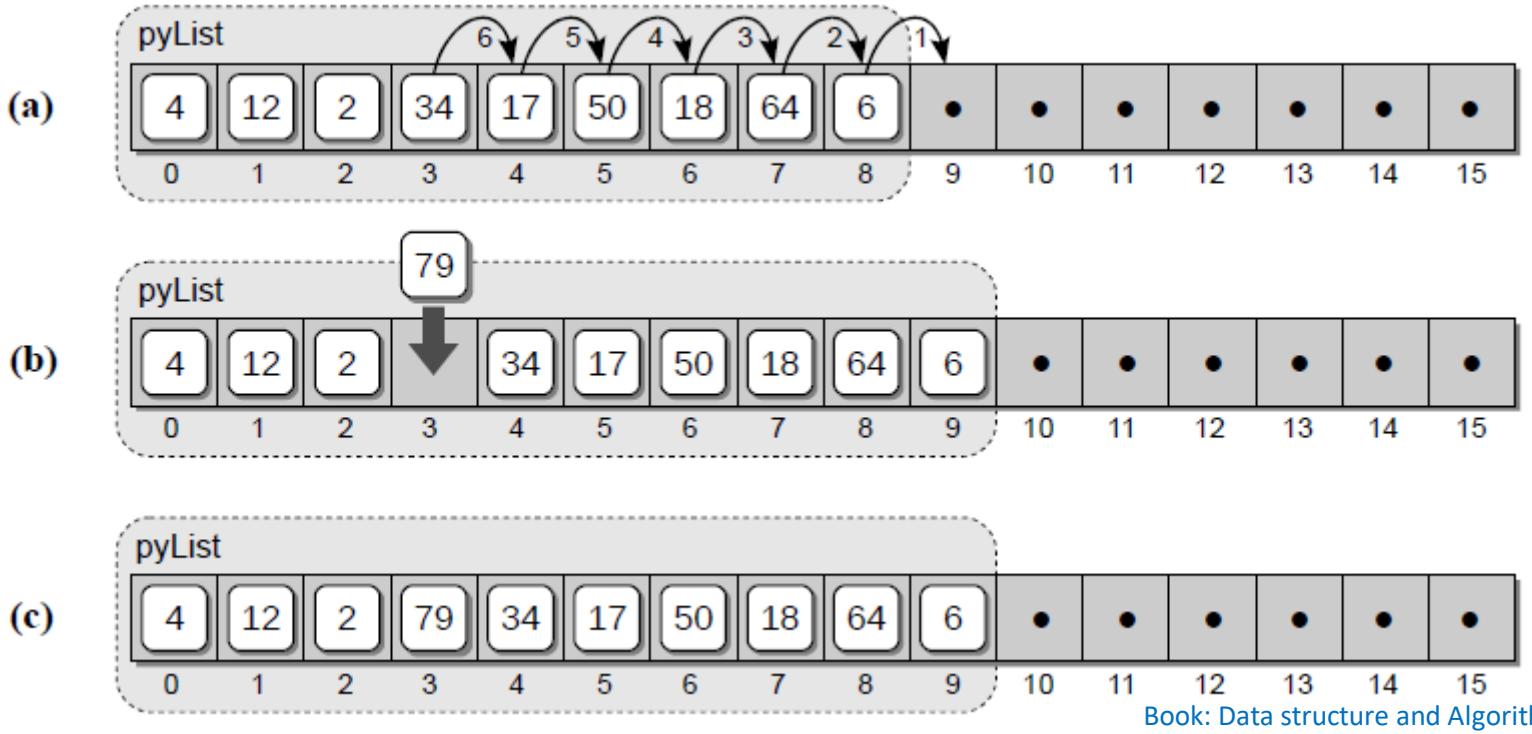
```
l.sort()
sorted(l)
```

- calling sort() **mutates** the list, returns nothing
- calling sorted() **does not mutate** list, must assign result to a variable

Lists

Note:

- Insert is computationally expensive compared to append because you need to shift elements internally to make room for new elements



Book: Data structure and Algorithms using Python, Rance D.Necaise, ISBN 978-0-470-61829-5.

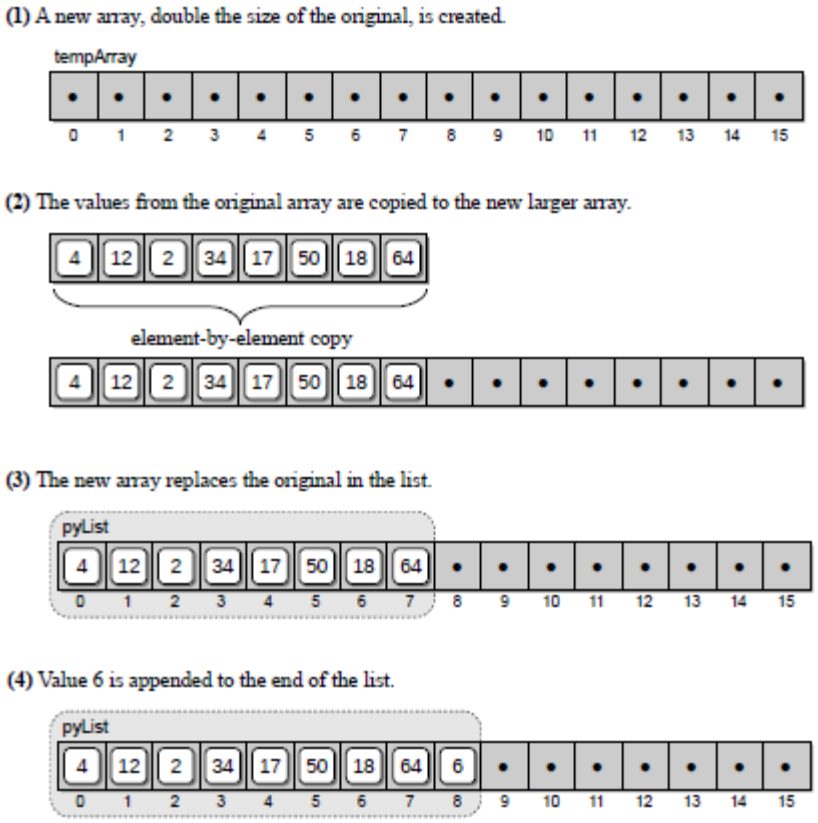
Lists

Note:

- List concatenation by addition is computationally expensive since a new list must be created and objects are copied → extend to append elements to an existing list is preferable.

Lists

Append a new item to the end of the list (if the array is full)?



Book: Data structure and Algorithms using Python, Rance D.Necaise, ISBN 978-0-470-61829-5.

Lists

Other methods

- sum: Adding up the elements of a list
- pop: modifies the list and returns the element that was removed. If you don't provide an index, it deletes and returns the last element.
- del: If you don't need the removed value
- remove: If you know the element you want to remove but not the index

```
numbers = [0,1,2,3,4]
l=sum(numbers)
l1 = numbers.pop(1)
del numbers[1]
numbers.remove(4)
```

```
10
1
[0, 3, 4]
[0, 3]
```

Lists

Lists and strings

- list: To convert from a string to a list of characters
- split: to break a string into words
- delimiter: specifies which characters to use as word boundaries
- join : puts a space between words

```
word = 'Python'
list1 = list(word)

sentence = 'I love Python'
delimiter = ' '
l = sentence.split(delimiter)

w = delimiter.join(l)
```

```
['P', 'y', 't', 'h', 'o', 'n']
['I', 'love', 'Python']
I love Python
```

Lists (Poll)

What will be the result for each print?

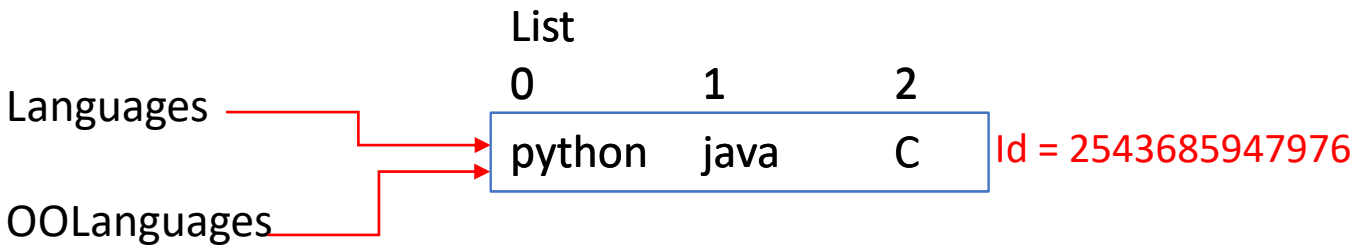
```
l = ['a', 'b', ['cc', 'dd', ['eee', 'fff']], 'g', 'h']

print(l[2][2])
A- eee
B- ['eee', 'fff']


print(l[-3][-1][-2])
A- eee
B- ['eee', 'fff']
```

List: Aliases

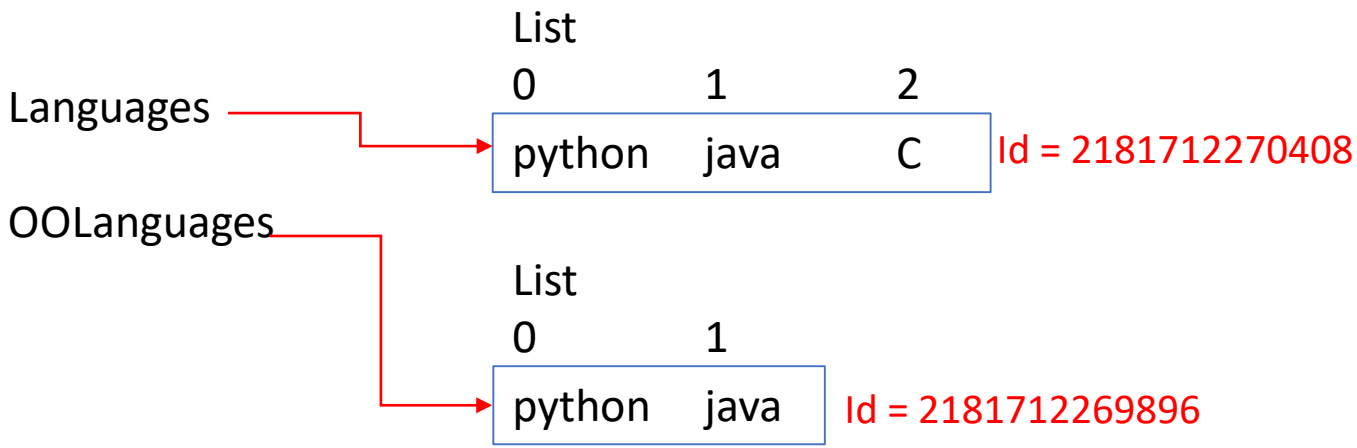
```
OOLanguages = ['python', 'java']
languages =  OOLanguages
languages.append('C')
print(languages)
print(OOLanguages)
```



OOLanguages is an **alias** for langauges: changing one changes the other!

List: Cloning

```
OOLanguages = ['python', 'java']
languages =  OOLanguages[:]
languages.append('C')
print(languages)
print(OOLanguages)
```



create a new list and **copy every element** using languages = OOLanguages[:]

Tuples

- an ordered sequence of elements, can mix element types
- represented with parentheses

```
t = ()  # empty tuple
t1 = tuple() # empty tuple
t = ('a’, 1, 'c', 'd', 'e’) # tuple with 5 elements
t2 = ('a’,) # extra comma means a tuple with one element
```

Tuples

- Tuple concatenation

```
(1,2,3) + (4,5)      # evaluates to (1,2,3,4,5)
```

- Tuple slices

```
t1=(1,'abc', 2.5, 'a')
t1[1:2]   ('abc',) -> extra comma means a tuple with one element
t1[1:3]   ('abc', 2.5)
```

Tuples

Tuples are immutable

```
t1=(1,'abc', 2.5, 'a')
t1[0]=2
```

TypeError: 'tuple' object does not support item assignment

Tuples

Tuples operators

- + : concatenation
- * : Repetition
- in : Returns true if an item exists in the given tuple
- not in : Returns true if an item does not exist in the given tuple

Tuples

Tuples methods

- len() : number of elements in the tuple
- max() (min()) : if the tuple contains numbers, it will return the highest (smallest) number. If it contains characters, it will return the one that comes last (first) in alphabetic order

Tuples

- Conveniently used to swap variable value

```
temp = x
x = y
y = temp
```

```
(x, y) = (y, x)
```

- Used to return more than one value from a function

```
def quotientRemainder(x, y):
        q = x //y
        r = x % y
        return (q, r)
```

Sets

- The set is a Python implementation of the set in Mathematics.

- A set object contains one or more items, not necessarily of the same type, which are separated by comma and enclosed in curly brackets {}.

```
set = {value1, value2, value3,...valueN}
```

Sets

A set doesn't store duplicate objects (Even if an object is added more than once inside the curly brackets, only one copy is held in the set object).

```
S1={1, 2, 2, 3, 4, 4, 5, 5}
print(S1)
```

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

Sets

- Sets are unordered.

- A set itself may be modified, but the elements contained in the set must be of an immutable type.

Sets

set() function

```
s1=set("Python course")
print(s1)

s2=set([4,55,17])
print(s2)

s3=set((4,55,17))
print(s3)
```

```
{'o', 't', 'e', 'h', 'u', 'y', 'c', 's', 'n', ' ', 'P', 'r'}
{17, 4, 55}
{17, 4, 55}
```

Sets

Set operations:

- Union of two sets: s1|s2 or s1.union(s2)
- Intersection of two sets: s1&s2 or s1.intersection(s2)
- Difference of two sets: s1-s2 or s1.difference(s2)
- Symmetric Difference: s1^s2 or s1.symmetric_difference(s2)

Sets

Built-in Set Methods

```
#Adds a new element in the set object.
s1= {'python', 'java', 'C'}
s1.add('C++')
print(s1)

#Adds multiple items from a list or a tuple.
s1.update(['R', 'perl'])
print(s1)

#Creates a copy of the set object.
s2=s1.copy()
print(s2)
```

```
{'java', 'python', 'C++', 'C'}
{'C', 'java', 'python', 'R', 'perl', 'C++'}
{'R', 'C', 'java', 'python', 'perl', 'C++'}
```

Sets

Built-in Set Methods

```
#Removes the contents of set object and results in an empty set.
s2.clear()
print(s2)

#Returns a set after removing an item from it. No changes are done
if the item is not present.
s1.discard('C+')
print(s1)

#Returns a set after removing an item from it. Results in an error
if the item is not present.
s1.remove('C')
print(s1)
```

```
set()
{'C', 'java', 'python', 'R', 'perl', 'C++'}
{'java', 'python', 'R', 'perl', 'C++'}
```

Sets (Poll)

What is the result of this statement:

```
{'b', 'a', 'r'} & set('qux')
```

A- {'b', 'r', 'a’}
B- {'q', 'r', 'x', 'u', 'b', 'a’}
C- {}
D- Set()

Frozen Sets

Exactly like a set, except that a frozenset is immutable:

- You can perform non-modifying operations on a frozenset

Dictionaries

- A dictionary is like a list:
  - List: indices have to be integers
  - Dictionary: indices can be any type.

- Dictionary allows mapping between a set of indices (keys) and a set of values. Each key maps to a value.

Dictionaries

- The association of a key and a value is called a key-value pair.

```
eng2fr = dict()
eng2fr = {'one': 'un', 'two': 'deux', 'three': 'trois'}
print(eng2fr)
print (eng2fr['two'])
```

```
{'one': 'un', 'two': 'deux', 'three': 'trois'}
deux
```

Dictionaries

Some methods

- len: it returns the number of key-value pairs
- in: tells you whether something appears as a key in the dictionary
- values: returns the values as a list

```
len(eng2fr)
'one' in eng2fr
vals = eng2fr.values()
```

```
3
True
dict_values(['un', 'deux', 'trois'])
```

Dictionaries

Dictionary as a set of counters:

An advantage of the dictionary implementation is that we don't have to know ahead of time which letters appear in the string and we only have to make room for the letters that do appear.

```
def histogram(s):
    d = dict()
    for c in s:
        if c not in d:
            d[c] = 1
        else:
            d[c] += 1
    return d

h = histogram('brontosaurus')
print(h)
```

```
{'b': 1, 'r': 2, 'o': 2, 'n': 1, 't': 1, 's': 2, 'a': 1, 'u': 2}
```

Dictionaries

Looping and dictionaries

If you use a dictionary in a for statement, it traverses the keys of the dictionary.

```
def print_hist(h):
    for c in h:
        print (c, h[c])

h = histogram('Python')
print_hist(h)
```

```
P 1
y 1
t 1
h 1
o 1
n 1
```

Dictionaries (Poll)

What is the result of this statement:

```
d = {'foo': 100, 'bar': 200, 'baz': 300}
d['bar':'baz']
```

A- (200, 300)

B- 200

C- none of the above

Which data type to use in your program?

Arrays VS Lists

- Lists can hold homogeneous items → similar to arrays

- Fundamentally different in terms of the operations one can perform on them:
  - Arrays: operations can be performed on all its item individually
  - Lists: can not perform operations on all its item individually

Arrays VS Lists

- Arrays :
  - need to be declared
  - very useful when dealing with a large collection of homogeneous data types → arrays may be faster and uses less memory when compared to lists
  - Are great for numerical operations

- Lists:
  - provides a large set of operations for managing the items contained in the list (inserting, searching, removing, extracting a subset of items, and sorting). The array structure only provides a limited set of operations for accessing the individual elements.

Tuples VS lists

Tuples: are immutable

→ This might be useful in situations where you might to pass the control to someone else but you do not want them to manipulate data in your collection

Sets VS lists

Sets : are a collection of distinct (unique) objects.

→ These are useful to create lists that only hold unique values in the dataset.

→ It is an unordered collection but a mutable one, this is very helpful when going through a huge dataset.

Dictionary

Dictionaries: useful when you need something similar to a telephone book

→ Dictionaries contain key-value pairs instead of just single elements.

Next modules

Data types and data structures more generally in computer science!