Advanced Strings and Lists Processing

Introduction to Computer Programming (Python)

Week 5

Note: using Python 3.11

Vivatsathorn Thitasirivit

Rev. 1.0 (Course 1/2023)

https://vt.in.th

Outlines

Advanced Strings and Lists Processing

Python

Review: Strings and Lists

In Python, lists and strings have their similarity. Their properties make them interchangeable.
This is just an example on how both are related.

$$\begin{tikzcd}
\text{List} & \texttt{['abc', 'def', 'ghi']} \arrow[d, shift right=6ex, "\texttt{.join('-')}"'] \\
\text{String} & \texttt{'abc-def-ghi'} \arrow[u, shift right=6ex, "\texttt{.split('-')}"']
\end{tikzcd}$$

Python

Review: String Basics

Python's string is a line/array of character (สายอักขระ).
You can manipulate it in many ways. The string type in Python is 'str'.

```
# String creation
str1 = 'Hello, this is Python!'  # or double quotes "..."

# Element indexing
print(str1[0])

# Range slicing
print(str1[2:5])

# Start:Stop:Step
print(str1[0:-2:2])

# Reverse
print(str1[::-1])
```

Python

Review: List Basics

Python's list is a dynamic array data structure, can be manipulated in many ways: element appending, insertion removal, etc.

```
# List creation
list1 = [1, 1, 2, 3, 5, 8, 13, 21]

# Element indexing
print(list1[0])

# Range slicing
print(list1[2:5])

# Start:Stop:Step
print(list1[0:-2:2])

# Reverse
print(list1[::-1])
```

Programming Nomenclature

Functions vs Methods

A function and a method are two similar things that each perform the same tasks. However, there are some major differences between those two.

1. **Function** – self-contained block of code that performs specific task. May takes parameters and may returns value.
2. **Method** – a function that is associated to an object or a class or a type. It is usually used to access/modify the state of the object to which it is associated.

```
Python Built-in Functions

e.g., id(...), len(...), sum(...),
min(...), max(...), all(...),
any(...), ...
```

```
Python List Methods

e.g., <list>.append(...), ...
```

Python
Mutability

Python data structures can be categorized into 2 types by their mutability property:

1. **Mutable types** – allow changes of values or data in place without affecting object's identity.

   Examples: list, dictionary, set

2. **Immutable types** – does not allow any changes of values or data in place.

   Examples: int, float, string, tuple, frozen set

```
[Python Console]

>>> list1 = [1, 2, 3, 4]

>>> print(id(list1))
4368257600

>>> list1.append(9)

>>> print(id(list1))
4368257600
```

Same ID means same object

```
[Python Console]

>>> str1 = "hello world"

>>> print(id(str1))
4363058608

>>> str1 = str1.upper()

>>> print(id(str1))
4360956592
```

Different object!

Strings

Python: String Methods

String Formatting: Multi-string literal

Multi-string Syntax

```python
str1 = 'This is long string 1.' 'This is long string 2.' 'This is long string 3.'

# str1 = 'This is long string 1.' + 'This is long string 2.' + 'This is long string 3.'

print(str1)
```

Output

```
This is long string 1.This is long string 2.This is long string 3.
```

Python: String Methods

String Formatting: Multi-string literal

Multi-line Syntax

```
str1 = ('This is long string 1.'
        'This is long string 2.'
        'This is long string 3.')

print(str1)
```

Output

```
This is long string 1.This is long string 2.This is long string 3.
```

Python: String Methods

String Formatting: f-string (Format)

The Syntax

```
name = 'Som'
age = '19'
str1 = f'Hello, {name}. You are {age}.'  # or use capital F

print(str1)
```

Output

```
Hello, Som. You are 19.
```

Note: Python's f-string is introduced in Python 3.6. Earlier versions can't use f-string.

Python: String Methods

String Formatting: f-string (Format)

Multi-line f-string

```
name = 'Som'
age = '19'
str1 = (f'Hello, {name}.\n'
        f'You are {age}.')

print(str1)
```

Output

```
Hello, Som.
You are 19.
```

Python: String Methods

String Formatting: r-string (Raw)

The Syntax

```
str1 = 'Sentence 1\nSentence 2'

print(str1)
```

Output

```
Sentence 1
Sentence 2
```

```
str2 = r'Sentence 1\nSentence 2'

print(str2)
```

Output

```
Sentence 1\nSentence 2
```

Python: String Methods

String Formatting: rf-string (Raw+Format)

You can combine raw string and format string by using rf symbol.

Python: String Methods

String Formatting: Techniques

Floating point formatting

```
# Format floats

val = 3.145

print(f'{val:.2f}')  # 2 decimal points
print(f'{val:.6f}')  # 6 decimal points
```

Output

```
3.15
3.145000
```

Python: String Methods

String Formatting: Techniques

Percent formatting

```
# Format percentage

val = 1 / 7

print(f'{val}')
print(f'{val:.2%}')
print(f'{val:.0%}')
```

Output

```
0.14285714285714285
14.29%
14%
```

String Formatting: Techniques

Width formatting

```
val = 92950

print(f'{val:8d}')   # Fill spaces
print(f'{val:08d}')  # Fill zeroes
```

* d *after number is optional.*

Output

```
   92950
00092950
```

String Formatting: Techniques

Justification

```python
word = 'Books List'

print(f'{word:<40}')  # Justify Left @ 40
print(f'{word:^40}')  # Justify Center @ 40
print(f'{word:>40}')  # Justify Right @ 40
```

Output

```
Books List
               Books List
                              Books List
```

Python: String Methods

String Formatting: Techniques

Numeric Formats

```python
val = 3192

# Hexadecimal
print(f'{val:x}')
print(f'{val:X}')

# Octal
print(f'{val:o}')

# Binary
print(f'{val:b}')

# Scientific Notation
print(f'{val:e}')
print(f'{val:E}')
print(f'{val:.1E}')
```

Output

```
c78
C78
6170
110001111000
9
3.192000E+03
3.2E+03
```

More string formatting, by BrianAllan

https://cheatography.com/brianallan/cheat-sheets/python-f-strings-number-formatting/

Python: String Methods

String Formatting: .format(...)

The Syntax

```
print('{0} and {1}'.format('chicken', 'eggs'))
print('{1} and {0}'.format('chicken', 'eggs'))
print('{} and {}'.format('chicken', 'eggs'))
```

Output

```
chicken and eggs
eggs and chicken
chicken and eggs
```

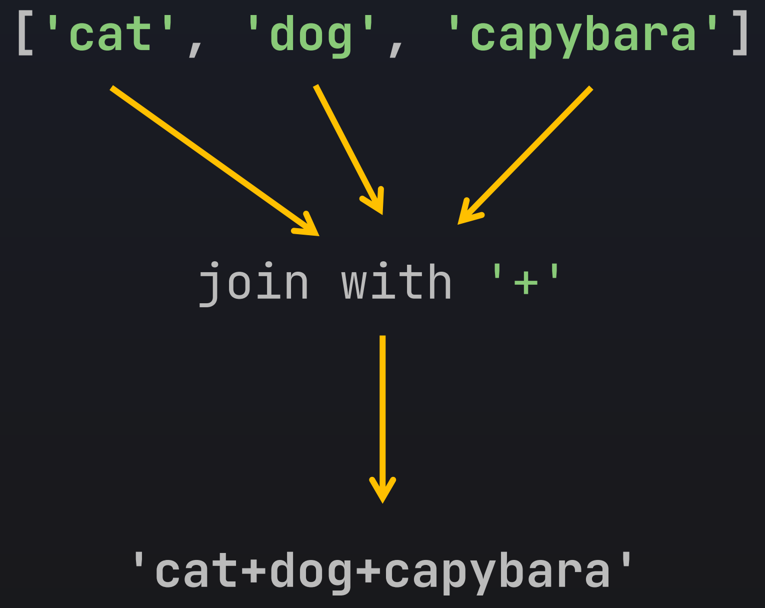
Python: String Methods

String Manipulation

Joining

Join elements in a list of strings by using the string as a delimiter for every string.

```python
list1 = ['cat', 'dog', 'capybara']

str1 = '+'.join(list1)

print(str1)
```



Python: String Methods

String Manipulation

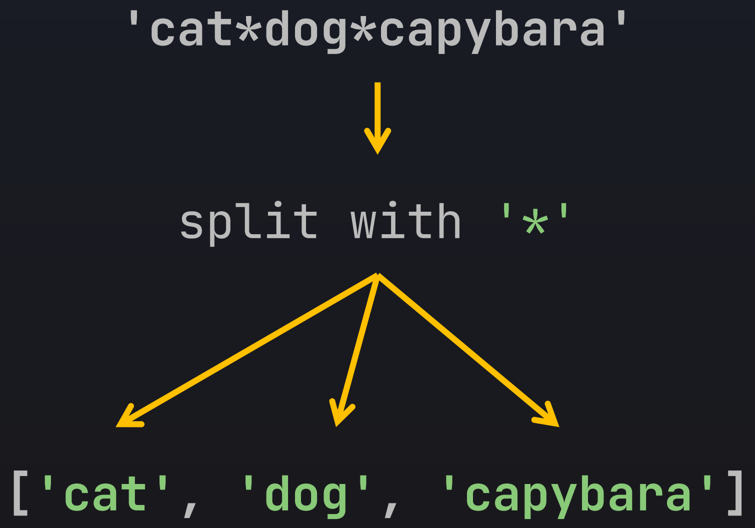
Splitting

Join elements in a list of strings by using the string as a delimiter for every string.

```
str1 = 'cat*dog*capybara'

list1 = str1.split('*')

print(list1)
```

You can also pass maximum number of splits as the second parameter.

There also exists `rsplit(...)` *method.*



Python: String Methods

String Manipulation

Splitting from lines

Split the string separated by Carriage Return (CR = \r) and/or Line Feed (LF = \n) automatically.

```python
str1 = 'cat\ndog\r\ncapybara\rmouse'

list1 = str1.splitlines()
list2 = str1.split('\r')

print(list1)
print(list2)
```

Output

```
['cat', 'dog', 'capybara', 'mouse']
['cat\ndog', '\ncapybara', 'mouse']
```

Python: String Methods

String Manipulation

Stripping a string

A string can be stripped.
(strip = remove spaces/characters before and/or after a string)

```python
str1 = ' \n\r   Hello World  \n \n \r  '

print(repr(str1))
print(repr(str1.strip()))
print(repr(str1.lstrip()))
print(repr(str1.rstrip()))
```

Output

```
' \n\r   Hello World  \n \n \r  '
'Hello World'
'Hello World  \n \n \r  '
' \n\r   Hello World'
```

By default, without passing any argument
to the method, it removes whitespaces,
carriage returns, and line feeds.

Python: String Methods

String Manipulation

Stripping a string (cont'd.)

Passing a string (characters to strip) argument.

```python
str1 = '.....X..Y..Z...'

print(repr(str1))
print(repr(str1.strip('.')))
print(repr(str1.lstrip('.')))
print(repr(str1.rstrip('.')))
```

Output

```
'.....X..Y..Z...'
'X..Y..Z'
'X..Y..Z...'
'.....X..Y..Z'
```

Python: String Methods

String Manipulation

Replacing substrings

Replace all or a certain number of occurrences of a substring with another substring.

```python
str1 = 'A wild cat runs after another cat.'

print(str1.replace('cat', 'fox'))
print(str1.replace(' ', '...'))
print(str1.replace(' ', '...', 3))
```

Output

```
A wild fox runs after another fox.

A...wild...cat...runs...after...another...cat.

A...wild...cat...runs after another cat.
```

Python: String Methods

More String Formatting

1. **Justify Left**	`.ljust(size)`
2. **Justify Center**	`.center(size)`
3. **Justify Right**	`.rjust(size)`
4. **Partition**	`.parititon(delimiter)` *and* `.rpartition(delimiter)`
5. **Zero fill**	`.zfill(size)`
6. **Expand tabs to space**	`.expandtabs(tabsize)`

```python
str1 = 'Word is from the God.'
str2 = '556'
str3 = 'a\tbyy\tcx'

print(str1.ljust(40))
print(str1.center(40))
print(str1.rjust(40))
print(str1.partition(' '))
print(str2.zfill(10))
print(str3.expandtabs(4))
```

```
[OUTPUT]



Word is from the God.
         Word is from the God.
                   Word is from the God.
('Word', ' ', 'is from the God.')
0000000556
a   byy cx
```

Python: String Methods

String Case Conversion

1. Uppercase	`.upper()`
2. Lowercase	`.lower()`
3. Title case	`.title()`
4. Capitalize	`.capitalize()`
5. Swap case	`.swapcase()`
6. Aggressive lowercase	`.casefold()`

```python
str1 = 'Word is from the God.'

print(str1.upper())
print(str1.lower())
print(str1.title())
print(str1.capitalize())
print(str1.swapcase())
print(str1.casefold())
```

```
[OUTPUT]

WORD IS FROM THE GOD.
word is from the god.
Word Is From The God.
Word is from the god.
wORD IS FROM THE gOD.
word is from the god.
```

Python: String Methods

String Testing

```
1.  .startswith(query)
2.  .endswith(query)
3.  .isalpha()
4.  .isascii()
5.  .isalnum()
6.  .isdecimal()
7.  .isdigit()
8.  .isnumeric()
9.  .isidentifier()
10. .isprintable()
11. .islower()
12. .isupper()
13. .istitle()
14. .isspace()
```

Python: String Methods

String Testing: string builtins

```
import string  # import Python's built-in string library

1.  string.ascii_letters       'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'

2.  string.ascii_lowercase     'abcdefghijklmnopqrstuvwxyz'

3.  string.ascii_uppercase     'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

4.  string.digits              '0123456789'

5.  string.hexdigits           '0123456789abcdefABCDEF'

6.  string.octdigits           '01234567'

7.  string.printable           '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~ \t\n\r\x0b\x0c'

8.  string.punctuation         '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'

9.  string.whitespace          ' \t\n\r\x0b\x0c'
```

Python: String Methods

String Searching

Membership 'in' Statement

Check if substring is present in the string or not.
The expression can be either `True` or `False`.

An empty string is always a substring of every string.

```
str1 = 'xyz'
str2 = 'hello world.'
str3 = 'I love xyz so much!'

print(str1 in str2)   # False
print(str1 in str3)   # True
print('' in str1)     # True
print('' in '')       # True
```

Python: String Methods
String Searching

Find and Index

You can check whether a substring is present in the string. You can also get its index within the string.

When the substring is not found, the behavior is different in these two methods.

1. `find` – returns -1 if not found.
2. `index` – raises ValueError Exception.

There also exists `rfind` *and* `rindex` to find.

```
str1 = 'xyz'
str2 = 'hello world.'
str3 = 'I love xyz and xyz so much!'

print(str3.find(str1))
print(str3.rfind(str1))
print(str3.index(str1))
print(str3.rindex(str1))

print(str2.find(str1))
print(str2.rfind(str1))
print(str2.index(str1))
print(str2.rindex(str1))
```

Lists

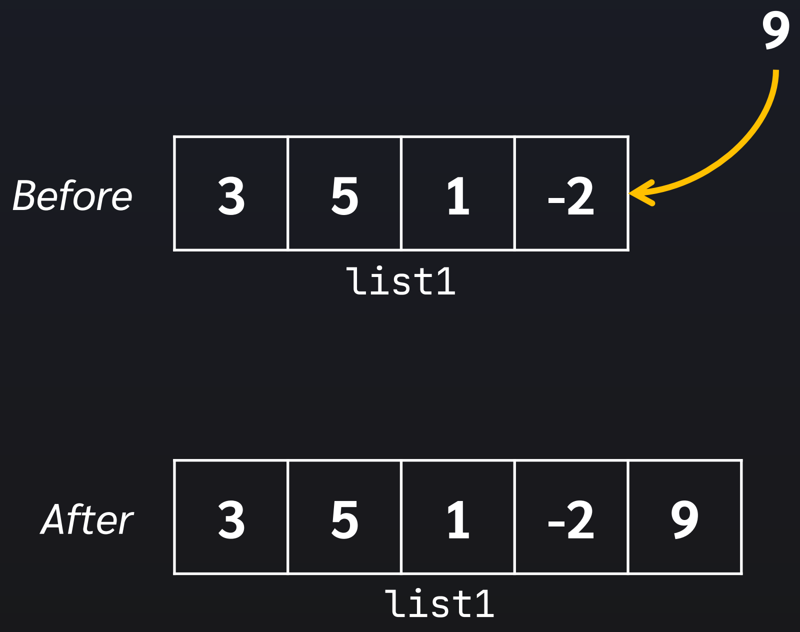
Python: List Methods

List Modification

Appending

Insert a new element to the back of the list.
The statement does not return anything.

```
list1 = [3, 5, 1, -2]

list1.append(9)

print(list1)  # [3, 5, 1, -2, 9]
```

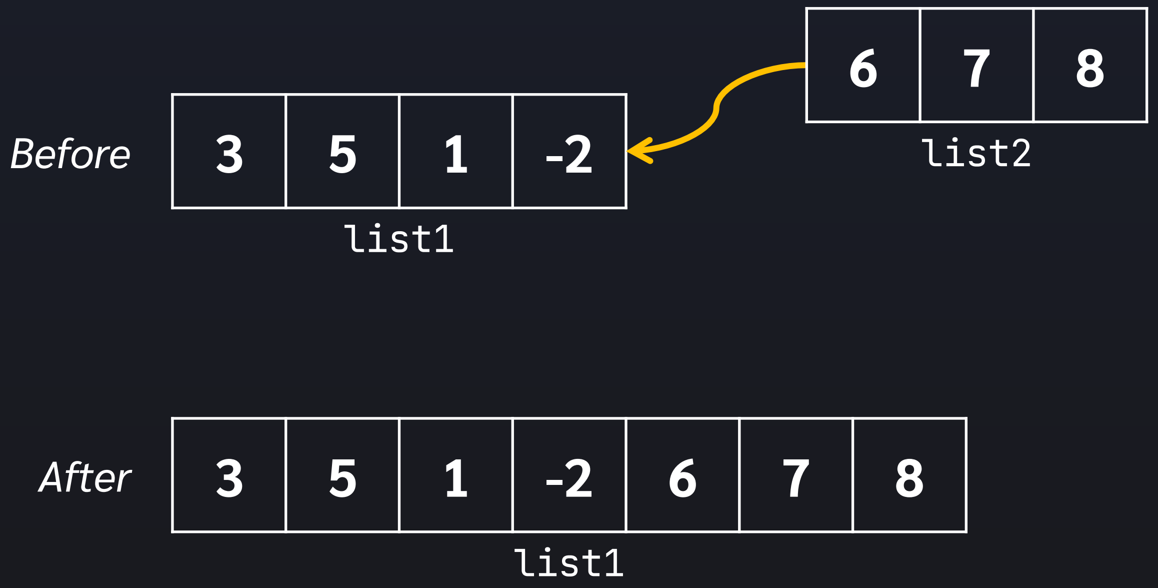


Python: List Methods

List Modification

Extending

Concatenate another list to current list.
The statement does not return anything.



```
list1 = [3, 5, 1, -2]
list2 = [6, 7, 8]

list1.extend(list2)

print(list1)
```

or

```
list1 = [3, 5, 1, -2]
list2 = [6, 7, 8]

list1 += list2  # list1 = list1 + list2

print(list1)
```

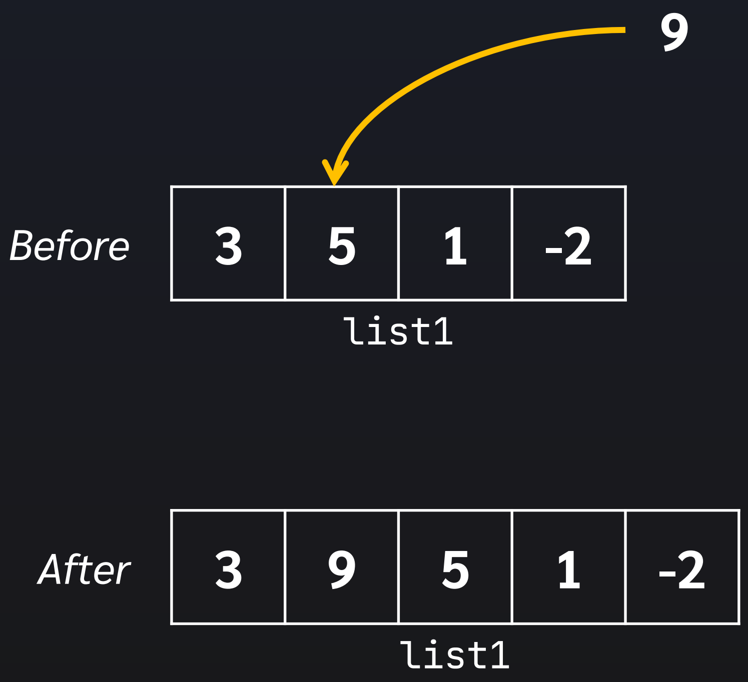
Python: List Methods

List Modification

Insertion

Insert an element at position (index) of the list.
The statement does not return anything.

```
list1 = [3, 5, 1, -2]

list1.insert(1, 9)

print(list1)  # [3, 9, 5, 1, -2]
```



Python: List Methods

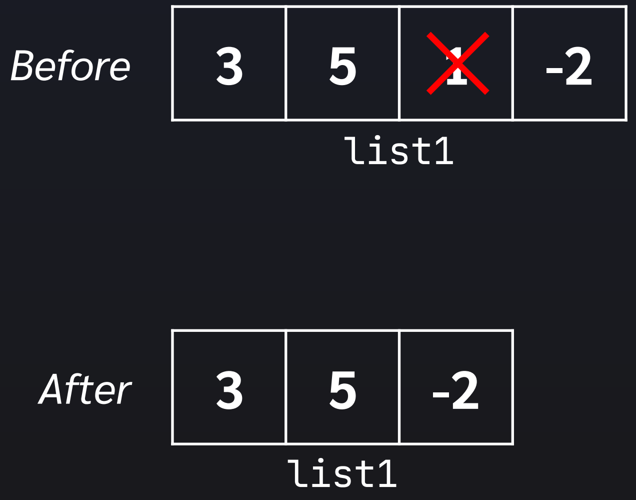
List Modification

Removal

Search and remove an element from the list.
The statement does not return anything.
The statement throws a *ValueError* exception if the list doesn't contain the value to remove.

```
list1 = [3, 5, 1, -2]

list1.remove(1)

print(list1)  # [3, 5, -2]
```

```
list1.remove(999)  # ValueError
```



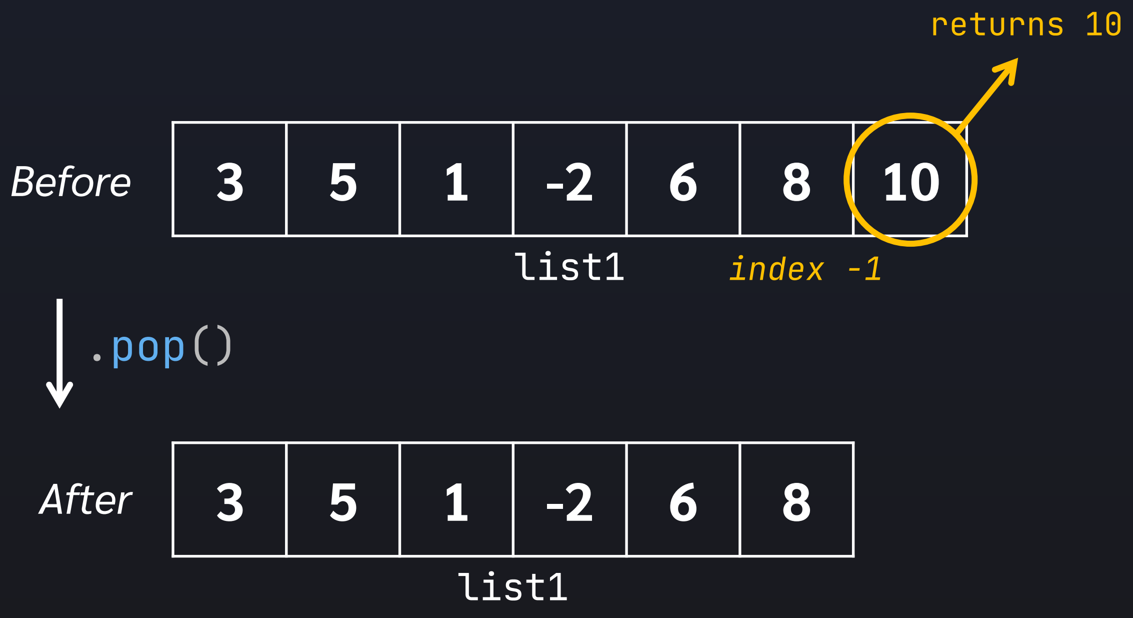
Python: List Methods

List Modification

Pop

Pop an element at index (default = -1 : last element) out of the list and returns element at that position.

```python
list1 = [3, 5, 1, -2, 6, 8, 10]

a = list1.pop()

print(list1)
print(a)

b = list1.pop(2)

print(list1)
print(b)
```



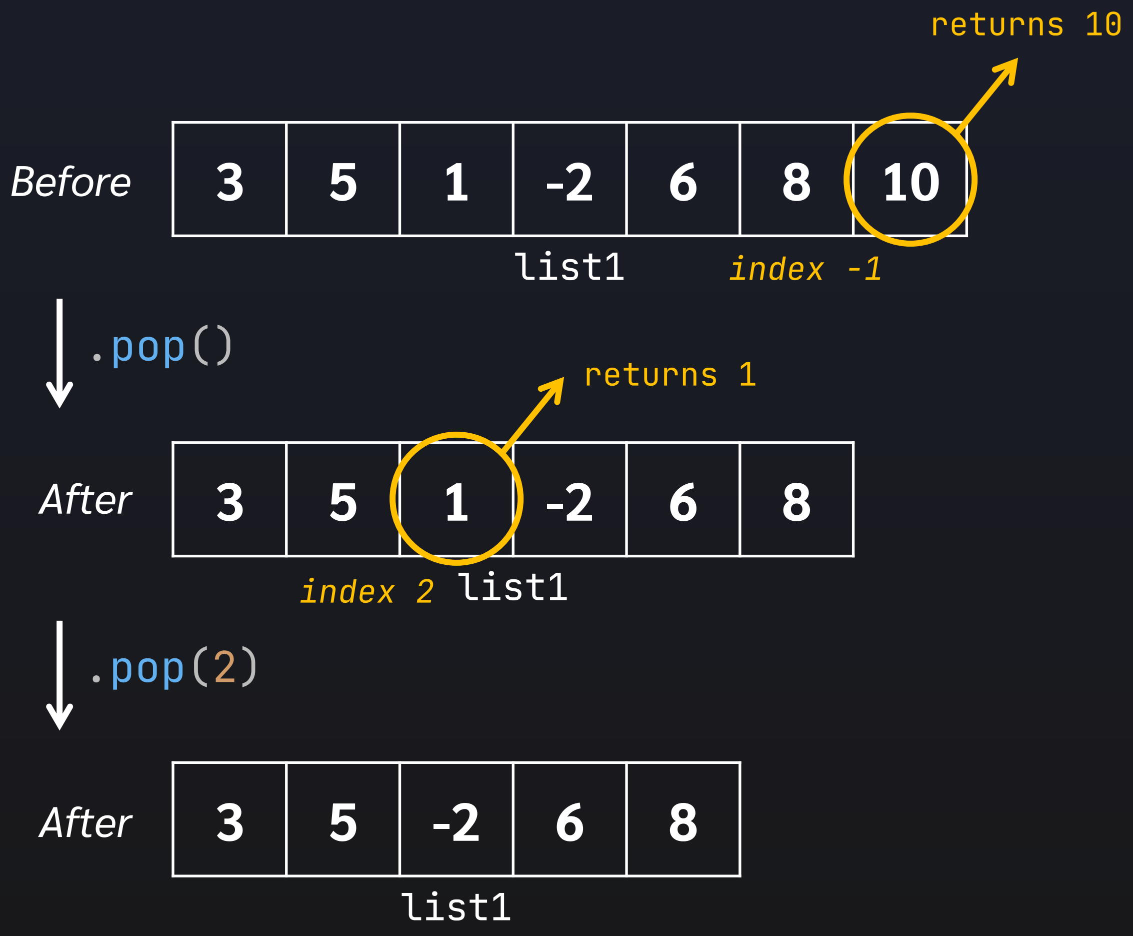
Python: List Methods

List Modification

Pop (cont'd.)

Pop an element at index (default = -1 : last element) out of the list and returns element at that position.

```
list1 = [3, 5, 1, -2, 6, 8, 10]

a = list1.pop()

print(list1)
print(a)

b = list1.pop(2)

print(list1)
print(b)
```



Python: List Methods

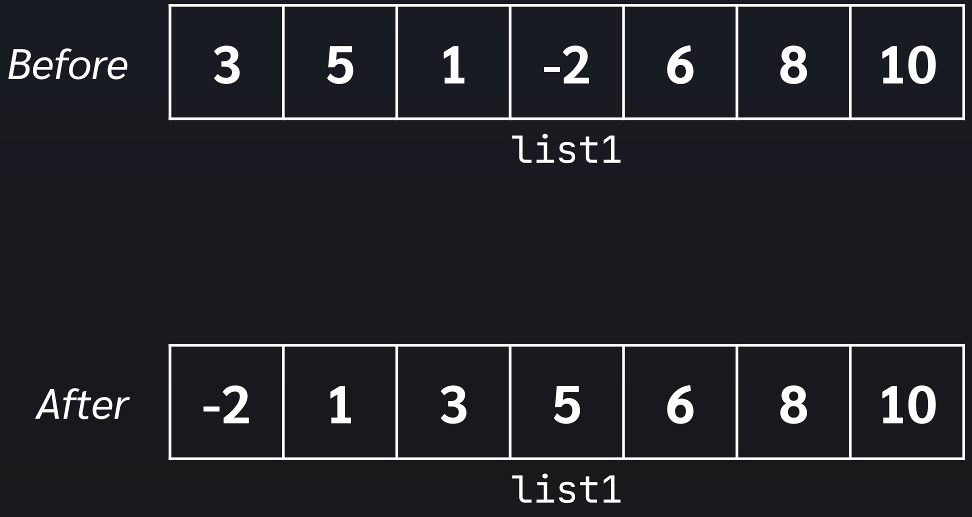
List Sorting and Reversing

Sorting (in-place)

Sort the current list in place.
The statement doesn't return anything.

```
list1 = [3, 5, 1, -2, 6, 8, 10]

list1.sort()

print(list1)
```

```
list1.sort(reverse=True)
```



Python: List Methods

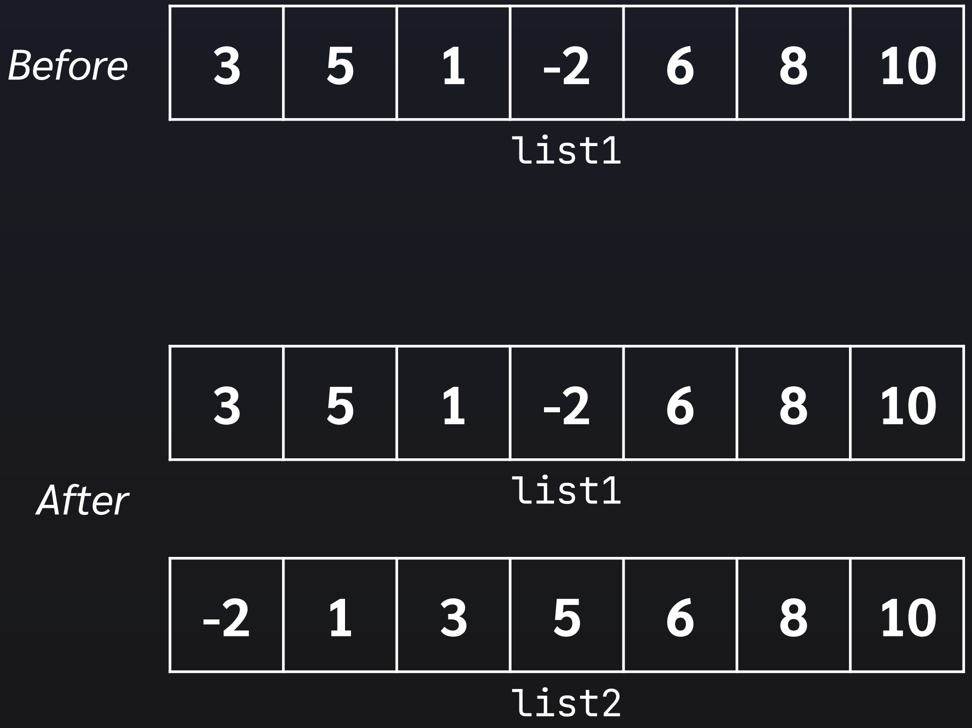
List Sorting and Reversing

Sorting (`sorted(...)` function)

Returns a new list with sorted element.
The function does not modify the list.

```
list1 = [3, 5, 1, -2, 6, 8, 10]

list2 = sorted(list1)

print(list1)
print(list2)
```

```
list2 = sorted(list1, reverse=True)
```



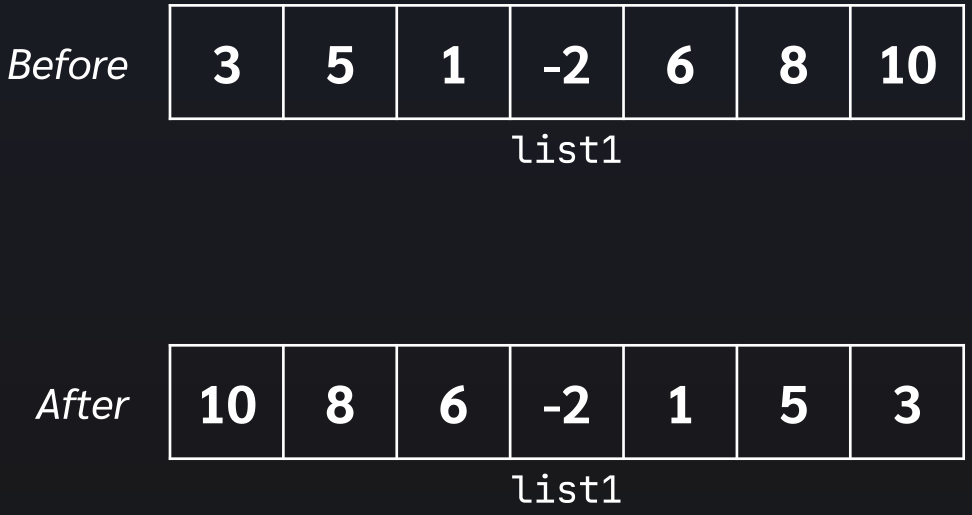
Python: List Methods

List Sorting and Reversing

Reversing (in-place)

Reverse the current list in place.
The statement doesn't return anything.

```
list1 = [3, 5, 1, -2, 6, 8, 10]

list1.reverse()

print(list1)
```



List Sorting and Reversing

Reversing (`reversed(...)` function)

Returns a reverse **iterator** of the list *(not a new list)*.
The function does not modify the list.

```
list1 = [3, 5, 1, -2, 6, 8, 10]

it = reversed(list1)

print(it)
print(list(it))
```

Python: List Methods

List Searching

Membership 'in' Statement

Check if an element is present in the list or not.
The expression can be either `True` or `False`.

```
list1 = [12, 56, 1, 99]

print(56 in list1)        # True
print(999 in list1)       # False
```

Python: List Methods
List Searching

Find (Index)

You can check whether an element is present in the list.
You can also get its index.

When the element is not found, it raises ValueError Exception.

```
list1 = [12, 56, 1, 99, 56]

print(list1.index(56))
print(list1.index(999))
```

Python: List Methods

List Searching

Counting occurrences

You can count occurrences of the element in the list.

```
list1 = [12, 56, 1, 99, 56]

print(list1.count(1))
print(list1.count(56))
print(list1.count(77))
```

```
1
2
0
```

Python: List Methods
List Searching

Min, Max, Sum, Any & All

You can count occurrences of the element in the list.

```
list1 = [12, 56, -1, 99, 56, 0, 7]

print(min(list1))  # -1
print(max(list1))  # 99
print(sum(list1))  # 229
print(any(list1))  # True
print(all(list1))  # False
```

Boolean Evaluation

Numbers:	False if equals 0.
String:	False if empty ('').
List:	False if empty ([ ]).
Bool:	As in literals

Python: List Methods

Multidimensional Lists

A multidimensional list is a form of n-dimensional array in Python. It is represented/constructed using nested list, or namely, a list of lists (2D), or in higher dimension.

It is used to represent a matrix/vector.

```
A = [[1, 2, 3],   # you can write it
     [4, 5, 6],   # all in one line
     [7, 8, 9],
     [10, 11, 12]]
```

```
# Access by Element
for row in A:
    for e in row:
        print(e, end=' ')  # each element in row
    print()  # new line after each row
```

```
# Access by Index: Able to reassign values
for i in range(len(A)):
    for j in range(len(A[i])):
        print(A[i][j], end=' ')  # each element in row
    print()  # new line after each row
```

Python: List Methods

Multidimensional Lists

A multidimensional list is a form of n-dimensional array in Python. It is represented/constructed using nested list, or namely, a list of lists (2D), or in higher dimension.

It is used to represent a matrix/vector.

```
A = [[1, 2, 3],  # you can write it
     [4, 5, 6],  # all in one line
     [7, 8, 9],
     [10, 11, 12]]
```

Python: List Methods

List Construction

From user input

You can construct a list from user's input by combination of functions and methods.

Constructing a vector

```python
n = int(input())  # how many inputs
arr = []  # empty list

for i in range(n):
    # append new int from user
    arr.append(int(input()))

print(arr)
```

Python: List Methods

List Construction

Constructing a 2D matrix (1)

```
m = int(input())  # m
n = int(input())  # n
arr = []  # matrix to be filled

for i in range(m):
    row = []  # temporary row
    for j in range(n):

        # Append each element to row
        row.append(int(input()))

    # Append the row to matrix
    arr.append(row)

print(arr)
```

Constructing a 2D matrix (2)

```
m = int(input())  # m
arr = []  # matrix to be filled

for i in range(m):
    row = input().split()
    arr.append(row)

print(arr)
```

Advanced Topics

Python: Advanced Topics

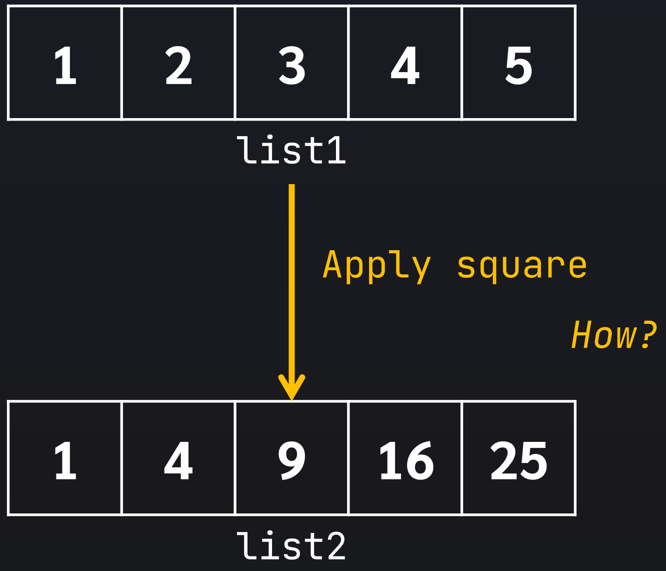
Functional Programming

Functional programming is a programming paradigm which functions are the core component of algorithms.

For example, you have a list of integers. You need to make a new list which each element is squared.

One way you can loop through the list and square each element, but there are better ways.

```python
list1 = [1, 2, 3, 4, 5]
list2 = []

for e in list1:
    list2.append(e ** 2)

print(list2)
```



Python: Advanced Topics

Functional Programming

Maps

A map in Python and other programming languages behaves and is defined similarly to a map in mathematics.

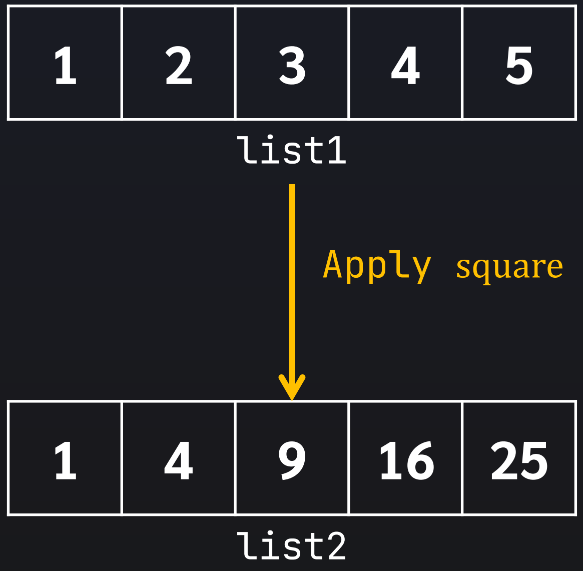
From the last example, we can apply a square function.
(A map is generalization of a function.)

We define a map square defined as follows.

$$\mathrm{square} : \mathbb{Z} \to \mathbb{Z} : \mathrm{square}(x) = x^2$$

The map square takes x and outputs x^2, i.e., a square function.

```python
def square(x):
    return x ** 2
```



Python: Advanced Topics

Functional Programming

Maps

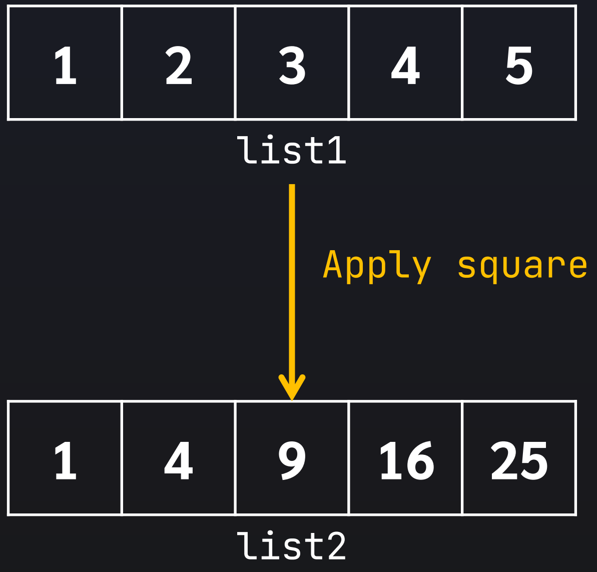
In Python, you can map a function to a list using `map` function.

Note: a `map` *returns a generator (like* `reversed(...)`*).*

```
def square(x):
    return x ** 2


list1 = [1, 2, 3, 4, 5]

mapped = map(square, list1)
list2 = list(mapped)

print(mapped)
print(list2)
```

Apply `square` *function*

Generate a list from map generator



Python: Advanced Topics

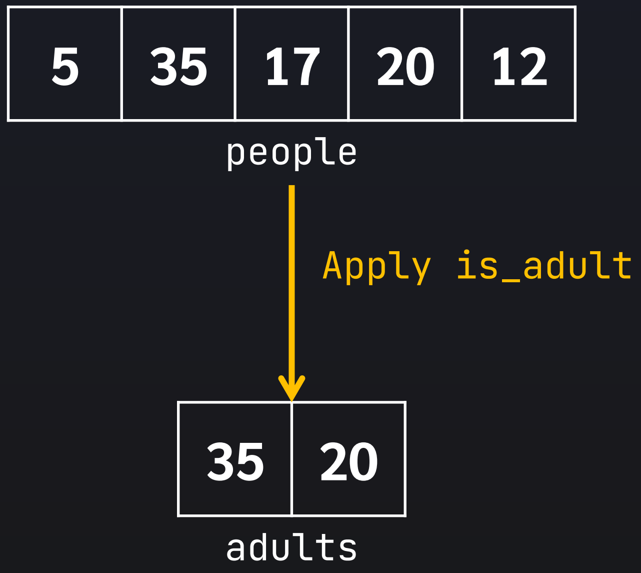
Functional Programming

Filters

In Python, A filter, like its name, is used to filter a list.

The criteria of the filter is defined by a function which then mapped to each element. Filtering to only 'True' elements.

```
def is_adult(age):
    return age >= 18


people = [5, 35, 17, 20, 12]

filtered = filter(is_adult, people)
adults = list(filtered)

print(filtered)
print(adults)
```



Note: a `filter` also returns a generator.

Python: Advanced Topics

Functional Programming

Reduce *(from built-in functools library)*

Python's Reduce has unique behaviors. It has similarity to Dynamic Programming concepts in some sense which starts from the base case and build up to the topmost value.

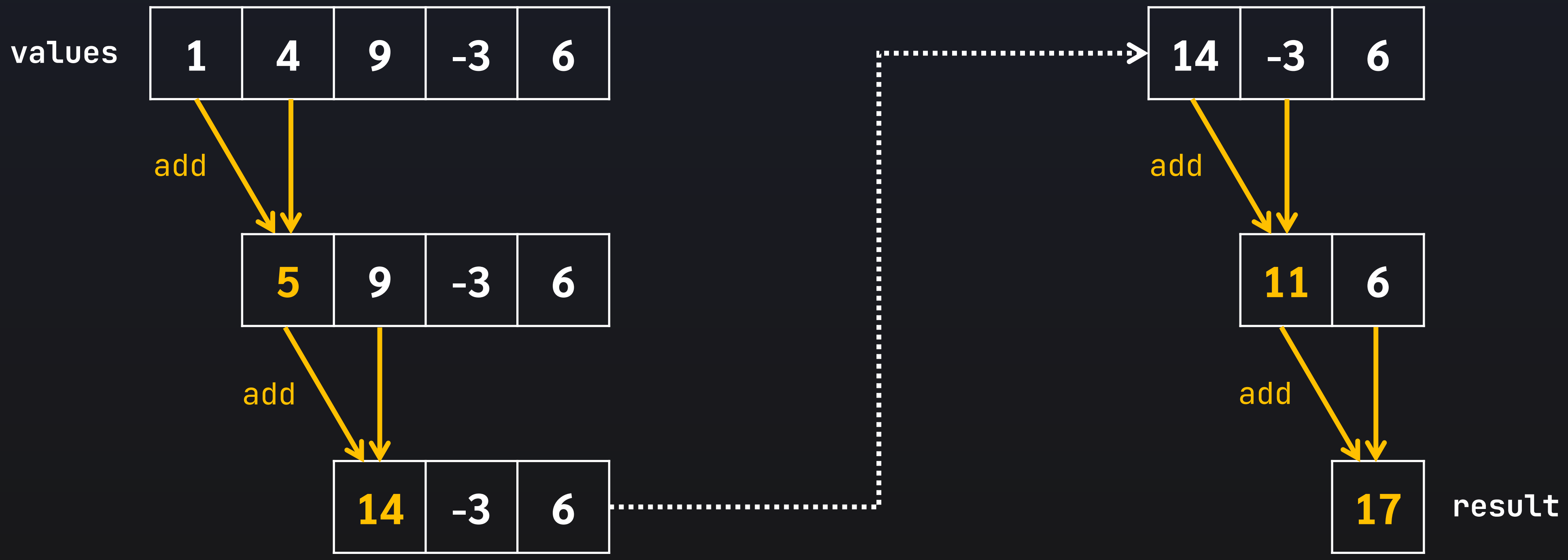
```python
from functools import reduce

def add(a, b):
    return a + b

values = [1, 4, 9, -3, 6]

result = reduce(add, values)

print(result)
```

Note: a `reduce` *also returns a generator.*

Python: Advanced Topics

Functional Programming

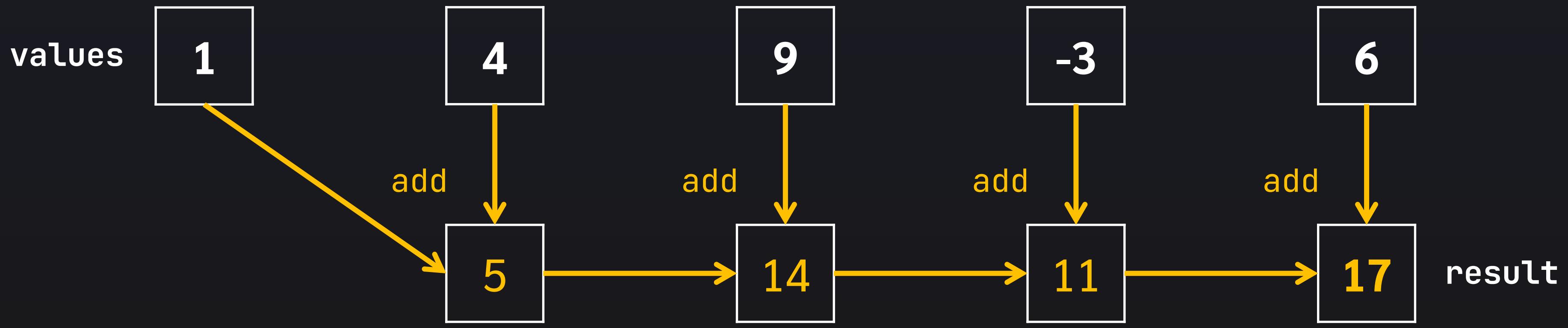
Reduce – Array Representation



Python: Advanced Topics

Functional Programming

Reduce – Tree Representation



Python: Advanced Topics

Functional Programming

Reduce – Function Call Representation

values	1	4	9	-3	6

```
result = add(add(add(add(1, 4), 9), -3), 6)
```

Python: Advanced Topics

Enumerate

Enumeration in Python is a way to loop through every element in a list while also count upward (with index). You can also set the start value (default start with 0).
This is a Pythonic way to iterate with index.

```python
list1 = [12, 56, 1, 99, 56]

for i, e in enumerate(list1):
    print(f'{i = } and {e = }')
```

Output

```
i = 0 and e = 12
i = 1 and e = 56
i = 2 and e = 1
i = 3 and e = 99
i = 4 and e = 56
```

Python: Advanced Topics

Zip

Many times, you might want to iterate through multiple lists simultaneously. One way you can use indexing, but what if the length of lists are not equal or when indexing is not available? There is more Pythonic approach: `zip` function.

```
list1 = [12, 56, 1, 99, 56]
list2 = [6, 7, 8, 13, 0]
list3 = [-1, 9, -8]
```

```
for i in range(min(len(list1), len(list2), len(list3))):
    x = list1[i]
    y = list2[i]
    z = list3[i]

    print(f'{x = } and {y = } and {z = }')
```

```
for x, y, z in zip(list1, list2, list3):
    print(f'{x = } and {y = } and {z = }')
```

Output

```
x = 12 and y = 6 and z = -1
x = 56 and y = 7 and z = 9
x = 1 and y = 8 and z = -8
```

Note: if the length are not equal, the minimum length is used.

Python: Advanced Topics

Iterators (Basic)

An iterator is an object that "iterates" through items in an "iterable" object, e.g., list, str.

To create a list iterator, call `iter(...)`

To get the next item, call `next(...)`

In "for e in ..." loop, the iterator is automatically created and called next until the end of iteration is reached.

At the end, the exception `StopIteration` is raised.

+ Iterator is lazy, meaning each element is lazily generated.

```
list1 = [12, 56, 1]

it = iter(list1)

print(it)

print(next(it))
print(next(it))
print(next(it))
print(next(it))
```

```
<list_iterator object at 0x105179d50>
12
56
1
Traceback (most recent call last):
    print(next(it))
          ^^^^^^^^
StopIteration
```

Generator Comprehension

A generator is a special type of iterator, which a new element will not be generated unless called.

You can create a generator function, but I will not cover in this topic.

However, you can create a simpler type of generator using **generator comprehension**.

Note: In Python, generator ⊆ iterator.

Note: Generators and Iterators may not be used in the same context.

Generator comprehension

```python
list1 = [6, 9, 3, 5, -7, 10]

gen = (e for e in list1)

print(gen)

print(next(gen))
print(next(gen))
print(next(gen))
print(next(gen))
print(next(gen))
print(next(gen))
print(next(gen))
```

Generate first 6 elements

raises StopIteration

Python: Advanced Topics

Generator Comprehension

Example,

Create a list generator which generates a square of each element.

One way, we can use map, but using generator comprehension makes the code more readable.

```
list1 = [6, 9, 3, 5, -7, 10]

gen = (e ** 2 for e in list1)

print(gen)

for e in gen:
    print(e)
```

```
gen = (e ** 2 for e in list1)
```

Python: Advanced Topics

List Comprehension

You can build a list by generator comprehension using **list comprehension** syntax. It is shorter and more Pythonic.

Example, construct a list where every element is a square of the given list.

```
list1 = [1, 2, 4, -5]

list2 = [e ** 2 for e in list1]

print(list2)
```

```
list2 = [e ** 2 for e in list1]
```

Python: Advanced Topics

Multiple Assignments

In Python, multiple assignment syntax can be used to increase the readability of code.

```python
a = 3
b = 2

print(a, b)
```

```python
a, b = 3, 2

print(a, b)
```

Python: Advanced Topics

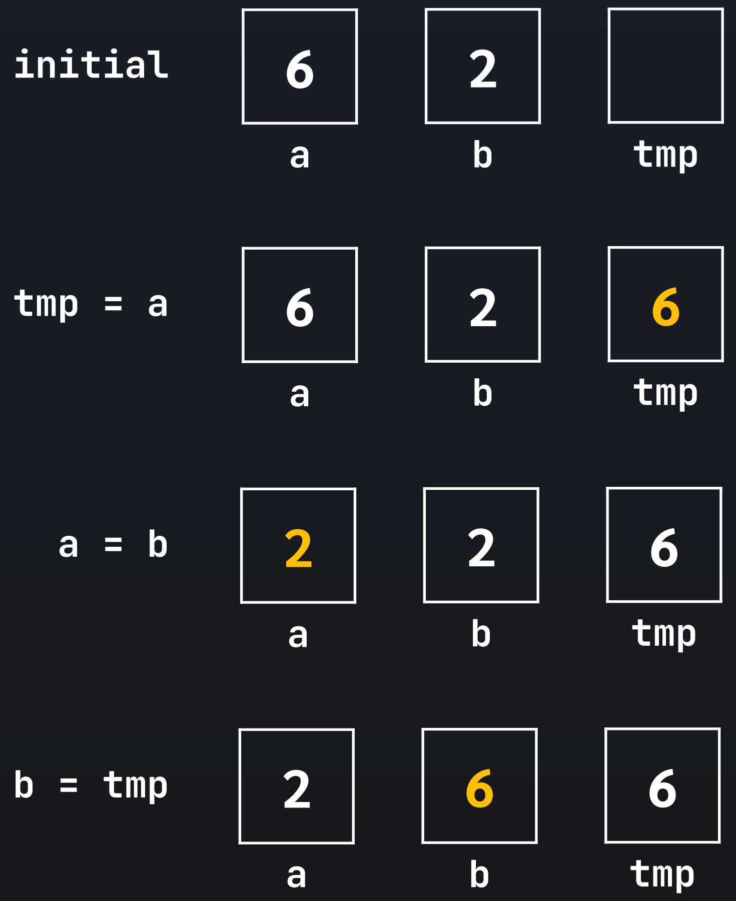
Multiple Assignments

Swapping 2 variables

```
a = 6
b = 2
```

```
tmp = a
a = b
b = tmp
```

```
print(a, b)
```



Python: Advanced Topics

Multiple Assignments

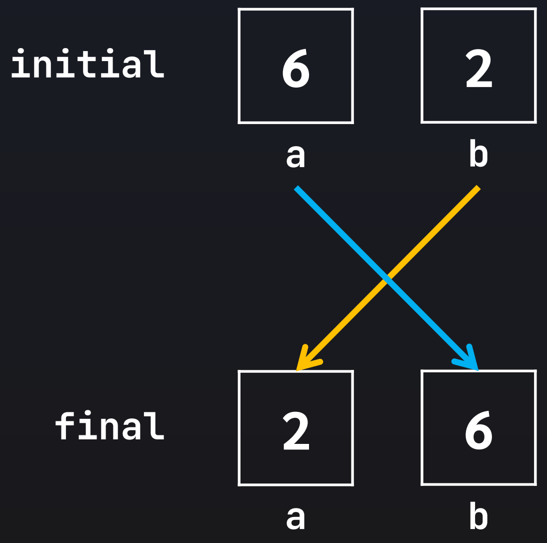
Swapping 2 variables

```
a = 6
b = 2
```

```
a, b = b, a
```

```
tmp = a
a = b
b = tmp
```

```
print(a, b)
```


initial
6
2
a
b
final
2
6
a
b