Py2: Bases of programming in Python

Lecturer: Clelia de Mulatier,

TAs: Digvijay, Barsha Bhattacharjee

Built-in data structure | **Loops and complexity** | **Functions**

General comments

Assignments. How did it go?

Please try to pass homework exercises during tutorials, as much as possible!

Plan. 0. Feedback on last week's exercises

1. Built-in data structures
2. Loop and notions of algorithmic complexity
3. Functions

Books.

A Whirlwind Tour of Python, by Jake VanderPlas (O'Reilly). Available online **here**.

0 -

This week's exercises

This week's exercises

H1: Information here

on how floating point numbers are encoded and on **subnormal number**

H3: Be careful with the units! Advices:

— write down units as comments near the definition of variables and near the functions
— Check unites quantities, try to use appropriate units
— defines intermediate variables

Scientific notations:

— powers of 10? 1.5e4

— Print with scientific notation: —> see Tips section of "*Py1_Lab*"

H3: precision limit!

— $1 + u^2 = 1 + 1e\text{-}32$
—> what happens? How can you still compute the difference in Dx?

Sqrt with complex type: cmath.sqrt()

Lists | Tuples | Dictionaries | Sets

Part 1

Built-in Data Structure

List, Tuple, Dictionary, Set

Lists | Tuples | Dictionaries | Sets

Part 1

Built-in Data Structure

List, Tuple, Dictionary, Set

Type Name	Example	Description
list	`[1, 2, 3]`	Ordered collection
tuple	`(1, 2, 3)`	Immutable ordered collection
dict	`{'a':1, 'b':2, 'c':3}`	Unordered (key,value) mapping
set	`{1, 2, 3}`	Unordered collection of unique values

Lists | Tuples | Dictionaries | Sets

Part 1

Built-in Data Structure

List, Tuple, Dictionary, Set

Type Name	Example	Description
list	`[1, 2, 3]`	Ordered collection
tuple	`(1, 2, 3)`	Immutable ordered collection
dict	`{'a':1, 'b':2, 'c':3}`	Unordered (key,value) mapping
set	`{1, 2, 3}`	Unordered collection of unique values

ordered Access the element by indexing / slicing

Immutable Element can't be changed

Lists

>>> I.1.Examples

Type Name	Example	Description
list	[1, 2, 3]	Ordered collection
tuple	(1, 2, 3)	Immutable ordered collection
dict	{'a':1, 'b':2, 'c':3}	Unordered (key,value) mapping
set	{1, 2, 3}	Unordered collection of unique values

List: **ordered** and **mutable** collection of objects.

```
# define a list:
fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
```

Mixed types

```
# Example of a list mixing types:
L = [1, 'two', 3.14, [0, 3, 5]]
```

Lists

>>> I.1.Examples

Type Name	Example	Description
list	[1, 2, 3]	Ordered collection
tuple	(1, 2, 3)	Immutable ordered collection
dict	{'a':1, 'b':2, 'c':3}	Unordered (key,value) mapping
set	{1, 2, 3}	Unordered collection of unique values

List: ordered and **mutable** collection of objects.

```
# define a list:
fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
```

Indexing

Access an element:

```
# Indexing: access an element:
print(fruits[4])
```

Lists

>>> I.1.Examples

Type Name	Example	Description
list	[1, 2, 3]	Ordered collection
tuple	(1, 2, 3)	Immutable ordered collection
dict	{'a':1, 'b':2, 'c':3}	Unordered (key,value) mapping
set	{1, 2, 3}	Unordered collection of unique values

List: ordered and **mutable** collection of objects.

```
# define a list:
fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
```

0 1 2 3 4 5 6

Indexing

Access an element:

```
# Indexing: access an element:
print(fruits[4])
```

```
kiwi
```

Lists

>>> I.1.Examples

Type Name	Example	Description
list	[1, 2, 3]	Ordered collection
tuple	(1, 2, 3)	Immutable ordered collection
dict	{'a':1, 'b':2, 'c':3}	Unordered (key,value) mapping
set	{1, 2, 3}	Unordered collection of unique values

List: ordered and **mutable** collection of objects.

```
# define a list:
fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
```

0 1 2 3 4 5 6

Indexing

Access an element:

```
# Indexing: access an element:
print(fruits[-1])
```

Lists

>>> I.1.Examples

Type Name	Example	Description
list	[1, 2, 3]	Ordered collection
tuple	(1, 2, 3)	Immutable ordered collection
dict	{'a':1, 'b':2, 'c':3}	Unordered (key,value) mapping
set	{1, 2, 3}	Unordered collection of unique values

List: ordered and **mutable** collection of objects.

```
# define a list:
fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
```

0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

Indexing

Access an element:

```
# Indexing: access an element:
print(fruits[-1])
```

```
banana
```

Lists

>>> I.1.Examples

Lists

Type Name	Example	Description
list	[1, 2, 3]	Ordered collection
tuple	(1, 2, 3)	Immutable ordered collection
dict	{'a':1, 'b':2, 'c':3}	Unordered (key,value) mapping
set	{1, 2, 3}	Unordered collection of unique values

List: ordered and **mutable** collection of objects.

```
# define a list:
fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
```

0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

Indexing

```
# Indexing: access an element:
print(fruits[4])
print(fruits[-1])
```

Modify an element:

```
# Modify an element via indexing:
fruits[0]='ananas'
```

Lists

>>> I.1.Examples

Lists

Type Name	Example	Description
list	[1, 2, 3]	Ordered collection
tuple	(1, 2, 3)	Immutable ordered collection
dict	{'a':1, 'b':2, 'c':3}	Unordered (key,value) mapping
set	{1, 2, 3}	Unordered collection of unique values

List: ordered and **mutable** collection of objects.

```
# define a list:
fruits = ['ananas', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
```

0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

Indexing

```
# Indexing: access an element:
print(fruits[4])
print(fruits[-1])
```

Modify an element:

```
# Modify an element via indexing:
fruits[0]='ananas'
```

Lists

>>> I.1.Examples

Type Name	Example	Description
list	[1, 2, 3]	Ordered collection
tuple	(1, 2, 3)	Immutable ordered collection
dict	{'a':1, 'b':2, 'c':3}	Unordered (key,value) mapping
set	{1, 2, 3}	Unordered collection of unique values

List: ordered and **mutable** collection of objects.

```
# define a list:
fruits = ['ananas', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
```

0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

Indexing

```
# Indexing: access an element:
print(fruits[4])
print(fruits[-1])
```

Modify an element:

```
# Modify an element via indexing:
fruits[0]='ananas'
```

Slicing

```
# Slicing: access several elements:
fruits[2:5]
```

Access several elements:

Lists

>>> I.1.Examples

Type Name	Example	Description
list	[1, 2, 3]	Ordered collection
tuple	(1, 2, 3)	Immutable ordered collection
dict	{'a':1, 'b':2, 'c':3}	Unordered (key,value) mapping
set	{1, 2, 3}	Unordered collection of unique values

List: ordered and **mutable** collection of objects.

```
# define a list:
fruits = ['ananas', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
```

0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

Indexing

```
# Indexing: access an element:
print(fruits[4])
print(fruits[-1])
```

Modify an element:

```
# Modify an element via indexing:
fruits[0]='ananas'
```

Slicing

```
# Slicing: access several elements:
fruits[2:5]
```

5 is excluded!

Access several elements:

```
['pear', 'banana', 'kiwi']
```

Lists

>>> I.1.Examples

Type Name	Example	Description
list	[1, 2, 3]	Ordered collection
tuple	(1, 2, 3)	Immutable ordered collection
dict	{'a':1, 'b':2, 'c':3}	Unordered (key,value) mapping
set	{1, 2, 3}	Unordered collection of unique values

List: ordered and **mutable** collection of objects.

```
# define a list:
fruits = ['ananas', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
```

0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

Indexing

```
# Indexing: access an element:
print(fruits[4])
print(fruits[-1])
```

Modify an element:

```
# Modify an element via indexing:
fruits[0]='ananas'
```

Slicing

```
# Slicing: access several elements:
fruits[2:5]
```

Access several elements:

```
['pear', 'banana', 'kiwi']
```

```
fruits[::2]
```

Lists

Type Name	Example	Description
list	[1, 2, 3]	Ordered collection
tuple	(1, 2, 3)	Immutable ordered collection
dict	{'a':1, 'b':2, 'c':3}	Unordered (key,value) mapping
set	{1, 2, 3}	Unordered collection of unique values

List: ordered and **mutable** collection of objects.

```
# define a list:
fruits = ['ananas', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
```

0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

Indexing

```
# Indexing: access an element:
print(fruits[4])
print(fruits[-1])
```

Modify an element:

```
# Modify an element via indexing:
fruits[0]='ananas'
```

Slicing

Access several elements:

```
# Slicing: access several elements:
fruits[2:5]
```

```
['pear', 'banana', 'kiwi']
```

```
fruits[::2]  # equivalent to fruits[0:len(fruits):2]
```

```
['orange', 'pear', 'kiwi', 'banana']
```

```
fruits[::-1]
```

```
['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', 'orange']
```

Lists

Type Name	Example	Description
list	`[1, 2, 3]`	Ordered collection
tuple	`(1, 2, 3)`	Immutable ordered collection
dict	`{'a':1, 'b':2, 'c':3}`	Unordered (key,value) mapping
set	`{1, 2, 3}`	Unordered collection of unique values

List: ordered and **mutable** collection of objects.

```
# define a list:
fruits = ['ananas', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
```

0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

Indexing

```
# Indexing: access an element:
print(fruits[4])
print(fruits[-1])
```

Modify an element:

```
# Modify an element via indexing:
fruits[0]='ananas'
```

Slicing

```
# Slicing: access several elements:
fruits[2:5]
```

Operations

```
# The addition '+' concatenates lists:
veggies = ['aubergine', 'carrot', 'potato']
fruits + veggies
```

Lists

>>> I.1.Examples

Type Name	Example	Description
list	[1, 2, 3]	Ordered collection
tuple	(1, 2, 3)	Immutable ordered collection
dict	{'a':1, 'b':2, 'c':3}	Unordered (key,value) mapping
set	{1, 2, 3}	Unordered collection of unique values

List: ordered and **mutable** collection of objects.

```
# define a list:
fruits = ['ananas', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
```

0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

Indexing

```
# Indexing: access an element:
print(fruits[4])
print(fruits[-1])
```

Modify an element:

```
# Modify an element via indexing:
fruits[0]='ananas'
```

Slicing

```
# Slicing: access several elements:
fruits[2:5]
```

Operations

```
# The addition '+' concatenates lists:
veggies = ['aubergine', 'carrot', 'potato']
fruits + veggies
```

Membership

```
# Check membership:
'carrot' in fruits
'carrot' not in fruits
```

Lists

>>> I.1.Examples

Type Name	Example	Description
list	[1, 2, 3]	Ordered collection
tuple	(1, 2, 3)	Immutable ordered collection
dict	{'a':1, 'b':2, 'c':3}	Unordered (key,value) mapping
set	{1, 2, 3}	Unordered collection of unique values

List: ordered and mutable collection of objects.

```
# define a list:
fruits = ['ananas', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
```

0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

Indexing

```
# Indexing: access an element:
print(fruits[4])
print(fruits[-1])
```

Modify an element:

```
# Modify an element via indexing:
fruits[0]='ananas'
```

Slicing

```
# Slicing: access several elements:
fruits[2:5]
```

Operations

```
# The addition '+' concatenates lists:
veggies = ['aubergine', 'carrot', 'potato']
fruits + veggies
```

Membership

```
# Check membership:
'carrot' in fruits
'carrot' not in fruits
```

Methods

```
# Count how many times the element 'apple' occurs in the fruits list:
fruits.count('apple')
```

>>> Q1

Lists

Lists — Summary

Type Name	Example	Description
list	[1, 2, 3]	Ordered collection
tuple	(1, 2, 3)	Immutable ordered collection
dict	{'a':1, 'b':2, 'c':3}	Unordered (key,value) mapping
set	{1, 2, 3}	Unordered collection of unique values

List: ordered and mutable collection of objects.

```
# define a list:
fruits = ['ananas', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
```

Mixed types

```
# Example of a list mixing types:
L = [1, 'two', 3.14, [0, 3, 5]]
```

Indexing

```
# Indexing: access an element:
print(fruits[4])
print(fruits[-1])
```

Modify an element:

```
# Modify an element via indexing:
fruits[0]='ananas'
```

Slicing

```
# Slicing: access several elements:
fruits[2:5]
```

Operations

```
# The addition '+' concatenates lists:
veggies = ['aubergine', 'carrot', 'potato']
fruits + veggies
```

Membership

```
# Check membership:
'carrot' in fruits
'carrot' not in fruits
```

Methods

```
# Count how many times the element 'apple' occurs in the fruits list:
fruits.count('apple')
```

Small exercise!
Elements in a list are Mutable

>>> Q3

Question: Can you guess and explain the outputs of the following statements:

```
x = [1, 10, 3, 4, 5, 6, 7, 8]
y = x
y[1] = 2
```

```
print(x)        ???
```

Small exercise!

>>> Q3

Question: Can you guess and explain the outputs of the following statements:

```
x = [1, 10, 3, 4, 5, 6, 7, 8]
y = x
y[1] = 2
```

```
print(x)        ???
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

x points to a list of elements that are mutable

y points to the same list (y=x)

as the elements of a list are mutable y[1] modify the list where both x and y are pointing

Small exercise!

>>> Q3

Question: Can you guess and explain the outputs of the following statements:

```
x = [1, 10, 3, 4, 5, 6, 7, 8]
y = x
y[1] = 2
```

```
print(x)
```

???

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

```
x = 10
y = x
y = y + 20
```

```
print(x)
```

???

Small exercise!

>>> Q3

Question: Can you guess and explain the outputs of the following statements:

```
x = [1, 10, 3, 4, 5, 6, 7, 8]
y = x
y[1] = 2
```

```
print(x)        ???
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

```
x = 10
y = x
y = 20
```

```
print(x)        ???
```

```
10
```

x is a pointer to an immutable object (integer 10)
when doing y = x; y = 20; the value 10 in x is not changed;
instead a new object is created and y points to that new object

Tuples

Tuple

>>> I.2. examples

Type Name	Example	Description
list	`[1, 2, 3]`	Ordered collection
tuple	`(1, 2, 3)`	Immutable ordered collection
dict	`{'a':1, 'b':2, 'c':3}`	Unordered (key,value) mapping
set	`{1, 2, 3}`	Unordered collection of unique values

Tuple: **ordered** and **immutable** collection of objects.

Tuples are similar to lists, main difference is that elements are immutable.

Immutable: this means that once a tuple is created, its size and content can't be changed anymore

```
# define a tuple:
t = (1,2,3)
```

```
# like lists: access an element via indexing:
t[0]
```

```
# Unlike lists: moodifying a tuple is not allowed:
t[2] = 5
```

```
TypeError: 'tuple' object does not support item assignment
```

```
# convert a tuple to a list:
l=list(t)
```

>>> Q5

Tuple: example

Type Name	Example	Description
list	`[1, 2, 3]`	Ordered collection
tuple	`(1, 2, 3)`	Immutable ordered collection
dict	`{'a':1, 'b':2, 'c':3}`	Unordered (key,value) mapping
set	`{1, 2, 3}`	Unordered collection of unique values

Tuple: **ordered** and **immutable** collection of objects.

Use of Tuples: a common use is in function that returns multiple values

Ex. `as_integer_ratio()`

is a method of float that returns a numerator and a denominator as a tuple

```
x = 0.125
x.as_integer_ratio()
```

```
(1, 8)
```

>>> Q7

Tuple: example

Type Name	Example	Description
list	`[1, 2, 3]`	Ordered collection
tuple	`(1, 2, 3)`	Immutable ordered collection
dict	`{'a':1, 'b':2, 'c':3}`	Unordered (key,value) mapping
set	`{1, 2, 3}`	Unordered collection of unique values

Tuple: **ordered** and **immutable** collection of objects.

Use of Tuples: a common use is in function that returns multiple values

Ex. `as_integer_ratio()`

is a method of float that returns a numerator and a denominator as a tuple

```
x = 0.125
x.as_integer_ratio()
```

```
(1, 8)
```

```
numerator, denominator = x.as_integer_ratio()
```

>>> Q7

Dictionary

>>> I.3. examples

Type Name	Example	Description
list	`[1, 2, 3]`	Ordered collection
tuple	`(1, 2, 3)`	Immutable ordered collection
dict	`{'a':1, 'b':2, 'c':3}`	Unordered (key,value) mapping
set	`{1, 2, 3}`	Unordered collection of unique values

Dictionary: collection of **maps between keys and values**.

Elements are of the form pairs of `key:value`.

Elements are not ordered; the value of any item can be access with the key.

```
# define a dictionary :
shopping = {'carrot': 10, 'mango': 3}
```

```
{'carrot': 10, 'mango': 3}
```

```
# access a value via the indexing syntax using the key:
shopping['mango']
```

```
3
```

```
# set a new value via the indexing syntax using the key:
shopping['mango'] = 6
```

```
{'carrot': 10, 'mango': 6}
```

```
# add a new item (pair key:value) via the indexing syntax:
shopping['orange'] = 5
```

```
{'carrot': 10, 'mango': 6, 'orange': 5}
```

```
# check membership: check if 'orange' is in shopping:
'orange' in shopping
```

```
True
```

Set

Type Name	Example	Description
list	`[1, 2, 3]`	Ordered collection
tuple	`(1, 2, 3)`	Immutable ordered collection
dict	`{'a':1, 'b':2, 'c':3}`	Unordered (key,value) mapping
set	`{1, 2, 3}`	Unordered collection of unique values

Set: Unordered collections of unique items.

As for sets in mathematics, you can perform operations such as union, intersection, difference, and symmetric difference.

```
# define two sets:
primes = {2, 3, 5, 7}
odds = set(range(1,10,2))
```

Set

Type Name	Example	Description
list	`[1, 2, 3]`	Ordered collection
tuple	`(1, 2, 3)`	Immutable ordered collection
dict	`{'a':1, 'b':2, 'c':3}`	Unordered (key,value) mapping
set	`{1, 2, 3}`	Unordered collection of unique values

Set: Unordered collections of unique items.

As for sets in mathematics, you can perform operations such as union, intersection, difference, and symmetric difference.

```
# define two sets:
primes = {2, 3, 5, 7}
odds = set(range(1,10,2))
```

```
{2, 3, 5, 7}
{1, 3, 5, 7, 9}
```

```

primes | odds
```

Bitwise OR

Set

Type Name	Example	Description
list	`[1, 2, 3]`	Ordered collection
tuple	`(1, 2, 3)`	Immutable ordered collection
dict	`{'a':1, 'b':2, 'c':3}`	Unordered (key,value) mapping
set	`{1, 2, 3}`	Unordered collection of unique values

Set: Unordered collections of unique items.

As for sets in mathematics, you can perform operations such as union, intersection, difference, and symmetric difference.

```
# define two sets:
primes = {2, 3, 5, 7}
odds = set(range(1,10,2))
```

```
{2, 3, 5, 7}
{1, 3, 5, 7, 9}
```

```
# union: union of the two sets, primes and odds:
primes | odds
```

```
{1, 2, 3, 5, 7, 9}
```

Bitwise OR

```

primes & odds
```

Bitwise AND

Set

Type Name	Example	Description
list	`[1, 2, 3]`	Ordered collection
tuple	`(1, 2, 3)`	Immutable ordered collection
dict	`{'a':1, 'b':2, 'c':3}`	Unordered (key,value) mapping
set	`{1, 2, 3}`	Unordered collection of unique values

Set: Unordered collections of unique items.

As for sets in mathematics, you can perform operations such as union, intersection, difference, and symmetric difference.

```
# define two sets:
primes = {2, 3, 5, 7}
odds = set(range(1,10,2))
```

```
{2, 3, 5, 7}
{1, 3, 5, 7, 9}
```

```
# union: union of the two sets, primes and odds:
primes | odds
```

```
{1, 2, 3, 5, 7, 9}
```

Bitwise OR

```
# intersection: items appearing in both set
primes & odds
```

```
{3, 5, 7}
```

Bitwise AND

```
# difference: items in odds but not in primes
odds - primes
```

Set

Type Name	Example	Description
list	`[1, 2, 3]`	Ordered collection
tuple	`(1, 2, 3)`	Immutable ordered collection
dict	`{'a':1, 'b':2, 'c':3}`	Unordered (key,value) mapping
set	`{1, 2, 3}`	Unordered collection of unique values

Set: Unordered collections of unique items.

As for sets in mathematics, you can perform operations such as union, intersection, difference, and symmetric difference.

```
# define two sets:
primes = {2, 3, 5, 7}
odds = set(range(1,10,2))
```

```
{2, 3, 5, 7}
{1, 3, 5, 7, 9}
```

```
# union: union of the two sets, primes and odds:
primes | odds
```

```
{1, 2, 3, 5, 7, 9}
```

Bitwise OR

```
# intersection: items appearing in both set
primes & odds
```

```
{3, 5, 7}
```

Bitwise AND

```
odds - primes
```

```
{1, 9}
```

>>> Q9

Part 2

Loops and algorithmic complexity

List, Tuple, Dictionary, Set

For Loop

Way to repeatedly execute the same code statement.

In python: `for` loop can iterate over any object that is *iterable*

Ex. Lists, Strings, Tuples, Range, etc…

```
# iteration over the characters of a string:
for l in 'words':
    print(l)
```

```
w
o
r
d
s
```

```
# iteration over the elements of a list:
for word in ['cat', 'mouse', 'elephant']:
    print(word, len(word))
```

```
cat 3
mouse 5
elephant 8
```

For Loop

Way to repeatedly execute the same code statement.

In python: `for` loop can iterate over any object that is *iterable*

Ex. Lists, Strings, Tuples, Range, etc…

Range object: generates a sequence of numbers.

syntax similar to slicing for lists.

By convention, exclude the last element:

```
for i in range(5,10):
    print(i, end=' ')
```

```
5 6 7 8 9
```

By convention, start from 0:

```
for i in range(10):
    print(i, end=' ')  # print all on same line
```

```
0 1 2 3 4 5 6 7 8 9
```

By steps of 2:

```
list(range(0, 10, 2))
```

```
[0, 2, 4, 6, 8]
```

>>> Q10

while Loop

Iterates until some condition is met.

The argument of the `while` loop is evaluated as a boolean statement, and the loop is executed until the statement evaluates to `False`.

```
i = 0
while i < 10:
    print(i, end=' ')
    i += 1
```

```
0 1 2 3 4 5 6 7 8 9
```

break: comment

List comprehensions

Nice feature of Python that allows you to build list from loops in a compact way.

Ex.

```
L = []
for n in range(12):
    L.append(n ** 2)
L
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121]
```

The list comprehension equivalent is:

```
[n ** 2 for n in range(12)]
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121]
```

List comprehensions

Nice feature of Python, that allows you to build list from loops in a compact way.

Ex. **Multiple iterations:**

```
[(i, j) for i in range(2) for j in range(3)]
```

```
[(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2)]
```

List comprehensions

Nice feature of Python, that allows you to build list from loops in a compact way.

Ex. **Multiple iterations:**

```
[(i, j) for i in range(2) for j in range(3)]
```

```
[(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2)]
```

Note that the second for acts as the interior index:

```
L=[]
for i in range(2):
    for j in range(3):
            L.append((i,j))
```

```
[(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2)]
```

List comprehensions

Nice feature of Python, that allows you to build list from loops in a compact way.

Ex. **Conditionals on iterator:**

```
[i for i in range(20) if i % 3 > 0]
```

List comprehensions

Nice feature of Python, that allows you to build list from loops in a compact way.

Ex. **Conditionals on iterator:**

```
[i for i in range(20) if i % 3 > 0]
```

```
[1, 2, 4, 5, 7, 8, 10, 11, 13, 14, 16, 17, 19]
```

Works for lists, sets, tuples, dictionaries

Functions

Way of creating more readable and reusable code.

Syntax: using `def`

Ex. function returning a list of the first `N` values of the Fibonacci sequence:

$$F_0 = 0, \quad F_1 = 1,$$
$$F_n = F_{n-1} + F_{n-2} \qquad \text{for } n > 1.$$

```
def fibonacci(N):
    L = []
    a, b = 0, 1
    while len(L) < N:
        a, b = b, a + b
        L.append(a)
    return L

fibonacci(10)
```

```
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

Algorithmic **Complexity**

Factoriel(n):

```
fact=1
for i in range(1,n+1):
    fact=fact*i
print(fact)
```

Algorithmic **Complexity**

Factoriel(n):

```
fact=1
for i in range(1,n+1):
    fact=fact*i
print(fact)
```

O(n) simple operations

Algorithmic **Complexity**

Factoriel(n):

```
fact=1
for i in range(1,n+1):
    fact=fact*i
print(fact)
```

O(n) simple operations

N_choose_K (N, K):

```
def NchooseK(N,K):
    result=1
    # intermediate prints:
    #print(N, 'choose 0 =', result)
    for k in range(K):
        # intermediate prints to observe what's happening during the loop:
        # These prints should be removed from the final function.
        result=(result*(N-k))//(k+1)
        #print(N, 'choose', k+1, '=', result)
    return(result)
```

Algorithmic **Complexity**

Factoriel(n):

```
fact=1
for i in range(1,n+1):
    fact=fact*i
print(fact)
```

O(n) simple operations

N_choose_K (N, K):

```
def NchooseK(N,K):
    result=1
    # intermediate prints:
    #print(N, 'choose 0 =', result)
    for k in range(K):
        # intermediate prints to observe what's happening during the loop:
        # These prints should be removed from the final function.
        result=(result*(N-k))//(k+1)
        #print(N, 'choose', k+1, '=', result)
    return(result)
```

O(K) simple operations

```
def NchooseK_bis(N,K):
    if K > N//2:
        return NchooseK(N,N-K)
    else:
        return NchooseK(N,K)
```

Algorithmic **Complexity**

Factoriel(n):

```
fact=1
for i in range(1,n+1):
    fact=fact*i
print(fact)
```

O(n) simple operations

N_choose_K (N, K):

```
def NchooseK(N,K):
    result=1
    # intermediate prints:
    #print(N, 'choose 0 =', result)
    for k in range(K):
        # intermediate prints to observe what's happening during the loop:
        # These prints should be removed from the final function.
        result=(result*(N-k))//(k+1)
        #print(N, 'choose', k+1, '=', result)
    return(result)
```

O(K) simple operations

```
def NchooseK_bis(N,K):
    if K > N//2:
        return NchooseK(N,N-K)
    else:
        return NchooseK(N,K)
```

min(K, N-K) simple operations

Int32 and int64

```
def factFn(n):
    fact=1
    for i in range(1,n+1):
        fact=fact*i
    return fact

import numpy as np
n=12

print("\n", factFn(n))
print(" 12!=", np.prod(np.arange(1,n+1, dtype='int32')))
print("2^31=", 2**31)
```

```
n=13
print("\n", factFn(n))
print(" 13!=", np.prod(np.arange(1,n+1, dtype='int32'),dtype='int32'))
print("2^31=", 2**31)
```

```
 479001600
 12!= 479001600
2^31= 2147483648

 6227020800
 13!= 1932053504
2^31= 2147483648
```