Object-Oriented Programming with Python

Master 1 Geniomhe
Univ. Évry Paris-Saclay

massinissa.hamidi@univ-evry.fr

Poll:
What comes to your mind when you hear object-oriented programming?

Demo

Goals: why do you need to learn OOP?

- Becoming a better programmer by writing better code
  - by "*better*", we mean: elegant, robust, reusable, readable, maintainable, bug-free, etc.
- Writing more complex pieces of software
  - What is the most complex software, in terms of LOC, that you have written?
- Reading and understanding code that fuels famous ML libraries (notably)
  - major ML libraries, like PyTorch, Scikit-learn, Tensorflow, etc., are built using OOP principles
  - You can also become a contributor to open source libraries in general
- Writing software that can be shared with the community
  - open science is based on reproducible results

Software development is complex … time-consuming and error-prone

- Needs for modeling tools and software development methodologies, like *Waterfall*, *Agile*, etc.

- Needs for specification and verification tools

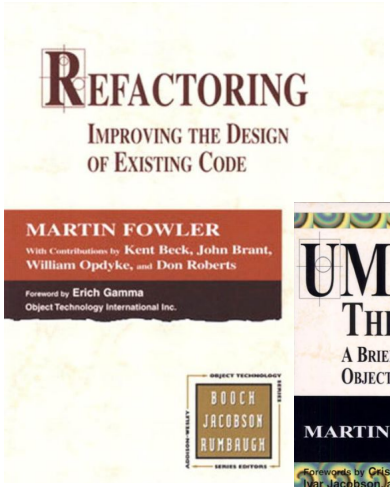
the destruction occurred due to a mismatch between the encoding memory registers (64-bit number coded into a 16-bit register)

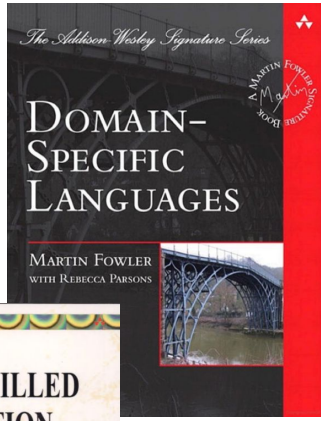
Instead of the metric units, the software team produced an output with non-SI units

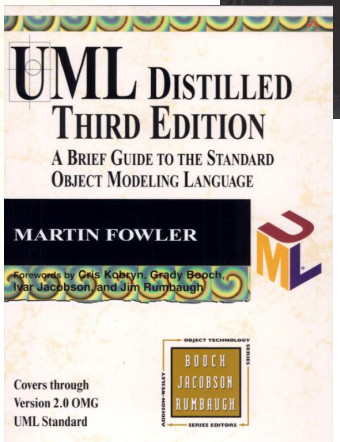
Remember the Mars Climate Orbiter incident from 1999?

Writing more elegant, readable, maintainable code

“Any fool can write code that a computer can understand.
Good programmers write code that humans can understand.” — Martin Fowler





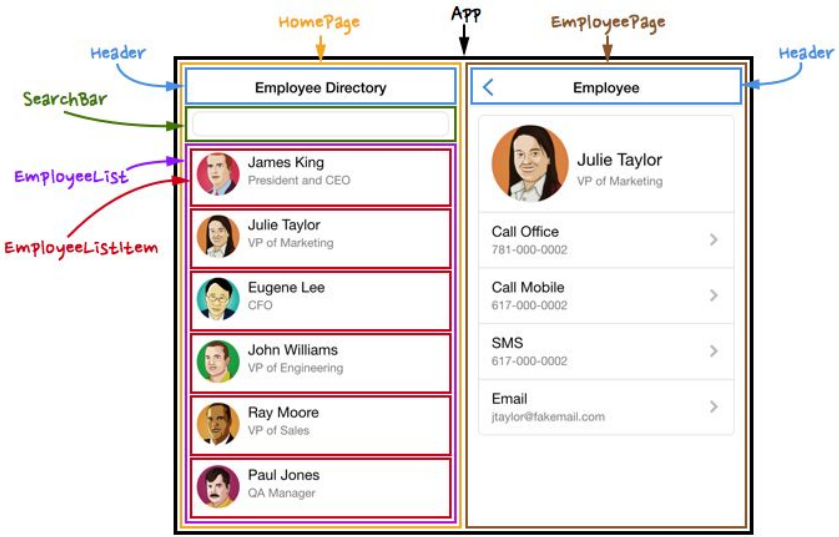


Reusability

The capability of a software component to be utilized within the same software system it was created for or to be integrated into external software systems.

Examples

- Functions
- Modules
- Software libraries
- Components (e.g., in React)



Reading and understanding code: PyTorch, Scikit-learn, etc.

https://github.com/pytorch/pytorch


```
class Linear(Module):
    r"""Applies an affine linear transformation to the incoming data: :math:`y = xA^T + b`.

    This module supports :ref:`TensorFloat32<tf32_on_ampere>`.

    On certain ROCm devices, when using float16 inputs this module will use :ref:`different preci

    Args:
        in_features: size of each input sample
        out_features: size of each output sample
        bias: If set to ``False``, the layer will not learn an additive bias.
            Default: ``True``

    Shape:
        - Input: :math:`(*, H_{in})` where :math:`*` means any number of
          dimensions including none and :math:`H_{in} = \text{in\_features}`.
        - Output: :math:`(*, H_{out})` where all but the last dimension
          are the same shape as the input and :math:`H_{out} = \text{out\_features}`.

    Attributes:
        weight: the learnable weights of the module of shape
            :math:`(\text{out\_features}, \text{in\_features})`. The values are
            initialized from :math:`\mathcal{U}(-\sqrt{k}, \sqrt{k})`, where
            :math:`k = \frac{1}{\text{in\_features}}`
        bias:   the learnable bias of the module of shape :math:`(\text{out\_features})`.
                If :attr:`bias` is ``True``, the values are initialized from
                :math:`\mathcal{U}(-\sqrt{k}, \sqrt{k})` where
                :math:`k = \frac{1}{\text{in\_features}}`

    Examples::

        >>> m = nn.Linear(20, 30)
        >>> input = torch.randn(128, 20)
        >>> output = m(input)
        >>> print(output.size())
```

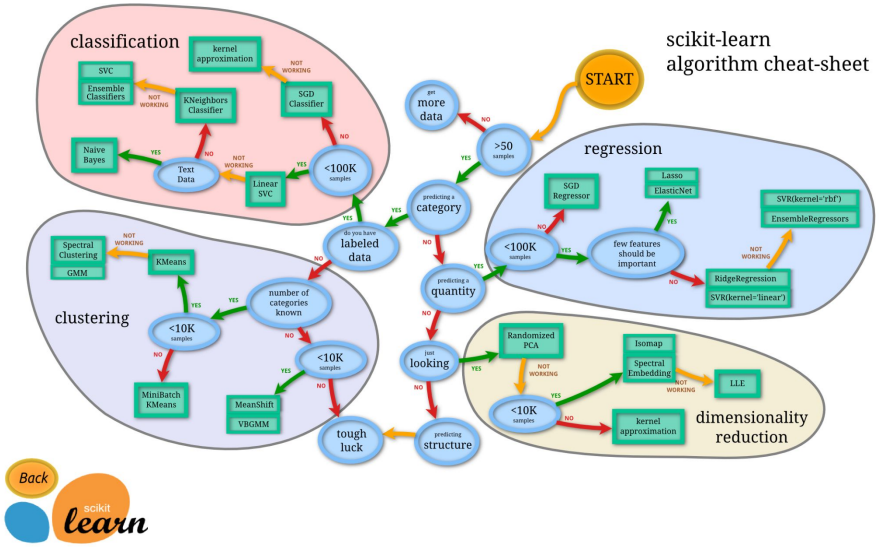
```
class Conv2d(_ConvNd):
    __doc__ = (
        r"""Applies a 2D convolution over an input signal composed of several input
    planes.

    In the simplest case, the output value of the layer with input size
    :math:`(N, C_{\text{in}}, H, W)` and output :math:`(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})`
    can be precisely described as:

    .. math::
        \text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) +
        \sum_{k = 0}^{C_{\text{in}} - 1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)


    where :math:`\star` is the valid 2D `cross-correlation`_ operator,
    :math:`N` is a batch size, :math:`C` denotes a number of channels,
    :math:`H` is a height of input planes in pixels, and :math:`W` is
    width in pixels.
    """
        + r"""

    This module supports :ref:`TensorFloat32<tf32_on_ampere>`.

    On certain ROCm devices, when using float16 inputs this module will use :ref:`different precision<fp16_on_

    * :attr:`stride` controls the stride for the cross-correlation, a single
      number or a tuple.

    * :attr:`padding` controls the amount of padding applied to the input. It
      can be either a string {{'valid', 'same'}} or an int / a tuple of ints giving the
      amount of implicit padding applied on both sides.
"""
        """
    * :attr:`dilation` controls the spacing between the kernel points; also
      known as the \u00e0 trous algorithm. It is harder to describe, but this `link`_
      has a nice visualization of what :attr:`dilation` does.
```

https://github.com/pytorch/pytorch/blob/db80b98ec460ca5b2fd84c1dfb6426925f64c8cc/torch/nn/modules/linear.py#L50
https://github.com/pytorch/pytorch/blob/db80b98ec460ca5b2fd84c1dfb6426925f64c8cc/torch/nn/modules/conv.py#L378

Reading and understanding code: PyTorch, Scikit-learn, etc.

```python
class ClusterMixin:
    """Mixin class for all cluster estimators in scikit-learn.

    - `_estimator_type` class attribute defaulting to `"clusterer"`;
    - `fit_predict` method returning the cluster labels associated to each sample.

    Examples
    --------
    >>> import numpy as np
    >>> from sklearn.base import BaseEstimator, ClusterMixin
    >>> class MyClusterer(ClusterMixin, BaseEstimator):
    ...     def fit(self, X, y=None):
    ...         self.labels_ = np.ones(shape=(len(X),), dtype=np.int64)
    ...         return self
    >>> X = [[1, 2], [2, 3], [3, 4]]
    >>> MyClusterer().fit_predict(X)
    array([1, 1, 1])
    """

    _estimator_type = "clusterer"

    def fit_predict(self, X, y=None, **kwargs):
        """
        Perform clustering on `X` and returns cluster labels.

        Parameters
        ----------
        X : array-like of shape (n_samples, n_features)
            Input data.

        y : Ignored
            Not used, present for API consistency by convention.

        **kwargs : dict
            Arguments to be passed to ``fit``.

            .. versionadded:: 1.4

        Returns
        -------
```

```python
class KMeans(_BaseKMeans):
    """K-Means clustering.

    Read more in the :ref:`User Guide <k_means>`.

    Parameters
    ----------

    n_clusters : int, default=8
        The number of clusters to form as well as the number of
        centroids to generate.

        For an example of how to choose an optimal value for `n_clusters` refer to
        :ref:`sphx_glr_auto_examples_cluster_plot_kmeans_silhouette_analysis.py`.

    init : {'k-means++', 'random'}, callable or array-like of shape \
            (n_clusters, n_features), default='k-means++'
        Method for initialization:

        * 'k-means++' : selects initial cluster centroids using sampling \
            based on an empirical probability distribution of the points' \
            contribution to the overall inertia. This technique speeds up \
            convergence. The algorithm implemented is "greedy k-means++". It \
            differs from the vanilla k-means++ by making several trials at \
            each sampling step and choosing the best centroid among them.

        * 'random': choose `n_clusters` observations (rows) at random from \
        data for the initial centroids.

        * If an array is passed, it should be of shape (n_clusters, n_features)\
        and gives the initial centers.

        * If a callable is passed, it should take arguments X, n_clusters and a\
        random state and return an initialization.

        For an example of how to use the different `init` strategy, see the example
        entitled :ref:`sphx_glr_auto_examples_cluster_plot_kmeans_digits.py`.
```



https://github.com/scikit-learn/scikit-learn

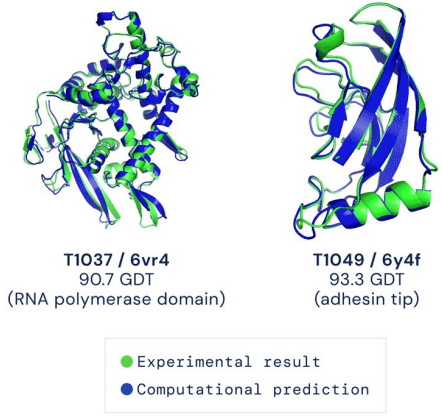
https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/base.py#L576
https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/cluster/_kmeans.py#L1190

Open science/reproducible research: AlphaFold

Highly accurate protein structure prediction with AlphaFold | Nature
https://github.com/google-deepmind/alphafold

```python
class FoldIteration(hk.Module):
  """A single iteration of the main structure module loop.

  Jumper et al. (2021) Suppl. Alg. 20 "StructureModule" lines 6-21

  First, each residue attends to all residues using InvariantPointAttention.
  Then, we apply transition layers to update the hidden representations.
  Finally, we use the hidden representations to produce an update to the
  affine of each residue.
  """

  def __init__(self, config, global_config,
               name='fold_iteration'):
    super().__init__(name=name)
    self.config = config
    self.global_config = global_config

  def __call__(self,
               activations,
               sequence_mask,
               update_affine,
               is_training,
               initial_act,
               safe_key=None,
               static_feat_2d=None,
               aatype=None):
    c = self.config

    if safe_key is None:
      safe_key = prng.SafeKey(hk.next_rng_key())

    def safe_dropout_fn(tensor, safe_key):
      return prng.safe_dropout(
          tensor=tensor,
          safe_key=safe_key,
          rate=c.dropout,
          is_deterministic=self.global_config.deterministic,
          is_training=is_training)

    affine = quat_affine.QuatAffine.from_tensor(activations['affine'])
```

```python
class RunModel:
  """Container for JAX model."""

  def __init__(self,
               config: ml_collections.ConfigDict,
               params: Optional[Mapping[str, Mapping[str, jax.Array]]] = None):
    self.config = config
    self.params = params
    self.multimer_mode = config.model.global_config.multimer_mode

    if self.multimer_mode:
      def _forward_fn(batch):
        model = modules_multimer.AlphaFold(self.config.model)
        return model(
            batch,
            is_training=False)
    else:
      def _forward_fn(batch):
        model = modules.AlphaFold(self.config.model)
        return model(
            batch,
            is_training=False,
            compute_loss=False,
            ensemble_representations=True)

    self.apply = jax.jit(hk.transform(_forward_fn).apply)
    self.init = jax.jit(hk.transform(_forward_fn).init)

  def init_params(self, feat: features.FeatureDict, random_seed: int = 0):
    """Initializes the model parameters.

    If none were provided when this class was instantiated then the parameters
    are randomly initialized.

    Args:
      feat: A dictionary of NumPy feature arrays as output by
        RunModel.process_features.
      random_seed: A random seed to use to initialize the parameters if none
        were set when this class was initialized.
    """
```



Course contents

1. Introduction, motivation, logistics, concrete example
2. Classes in UML
3. POO principles
4. Class, object, methods, attributes, etc.
5. Abstraction, Inheritance, Mixins
6. Python decorators, dunder functions (__init__, __call__, etc.)
7. Design patterns

Credits and Textbooks

- Yann Régis-Gianas, *POCA*, Univ. Paris Cité (Diderot)
- Delphine Longuet, *UML*, Polytech Paris-Saclay
- Martin Fowler et al., *Refactoring: Improving the Design of Existing Code*
- Eric Freeman and Elisabeth Freeman, *Head First Design Patterns*
- Erich Gamma et al., *Design Patterns: Elements of Reusable Object-Oriented Software*

Course logistics

Pedagogical team

- PI Massinissa Hamidi (massinissa.hamidi@univ-evry.fr) Office 211
- ~~TA Clément Bernard (clement.bernard@univ-evry.fr) Office 214~~

Evaluation

- Short MCQs at the beginning of the labs (20%)
- Project (40%)
- Exam (40%)

Project's principles

Goal: Build an ML library similar to scikit-learn. Make full use of OOP concepts learned during this course: abstraction, encapsulation, design patterns, etc.

Groups of 2-3 students?
Deliverables: Code+Report+Presentation
… more details to come.

Demo

Global variables/Shared state/Side effects

Side effects are operations that do more than return a result: mutate, I/O, exceptions, threads, etc.

⇒ Unpredictability of the state of the program

Global variables/Shared state/Side effects

Side effects are operations that do more than return a result: mutate, I/O, exceptions, threads, etc.

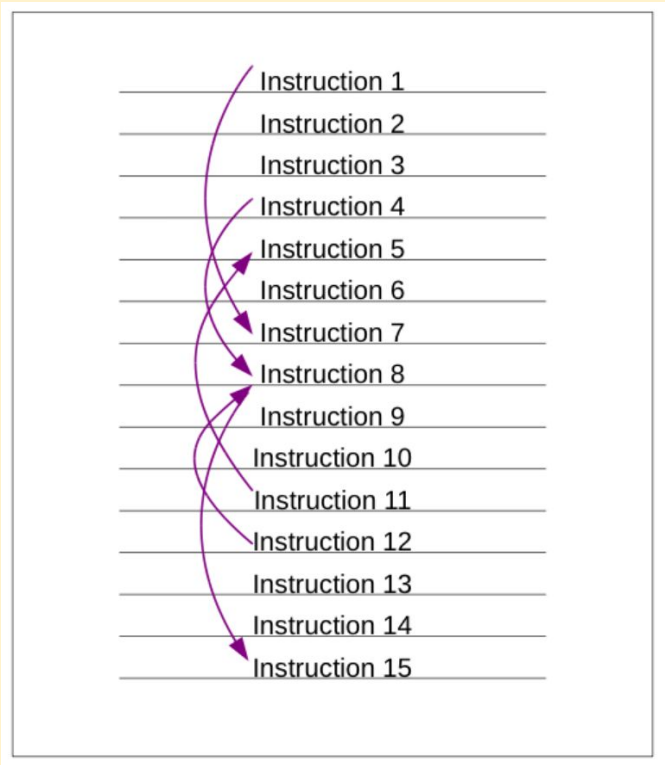
Example: functions that modify a global variable and perform I/O

```
15  x = 1
16
17  def add(a, b):
18      global x
19      x = x+1
20      return a+b
21
22  print(add(5,4))
23  print(x)
```

```
27  l = ['rr', 54, 'dual']
28
29  def multiply(a, b):
30      l[0] = 'ff'
31      return a*b
32
33  print(multiply(5, 5))
34  print(l)
```

```
48  def multiply(a, b):
49      b = input("b: ")
50      return a*b
51
52  print(multiply(5,5))
```

Non-structured programming

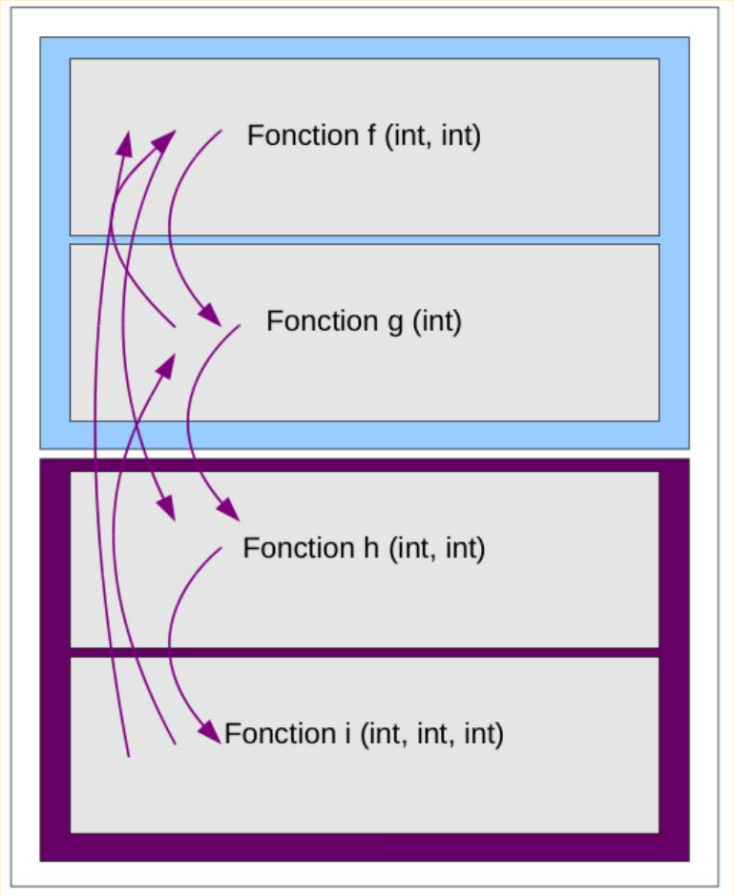



```
1 i=0
2 i=i+1
3 PRINT i
4 IF i>=100 THEN GOTO 6
5 GOTO 2
6 END
```

From YRG slides POCA

A ⇒ B means A references B

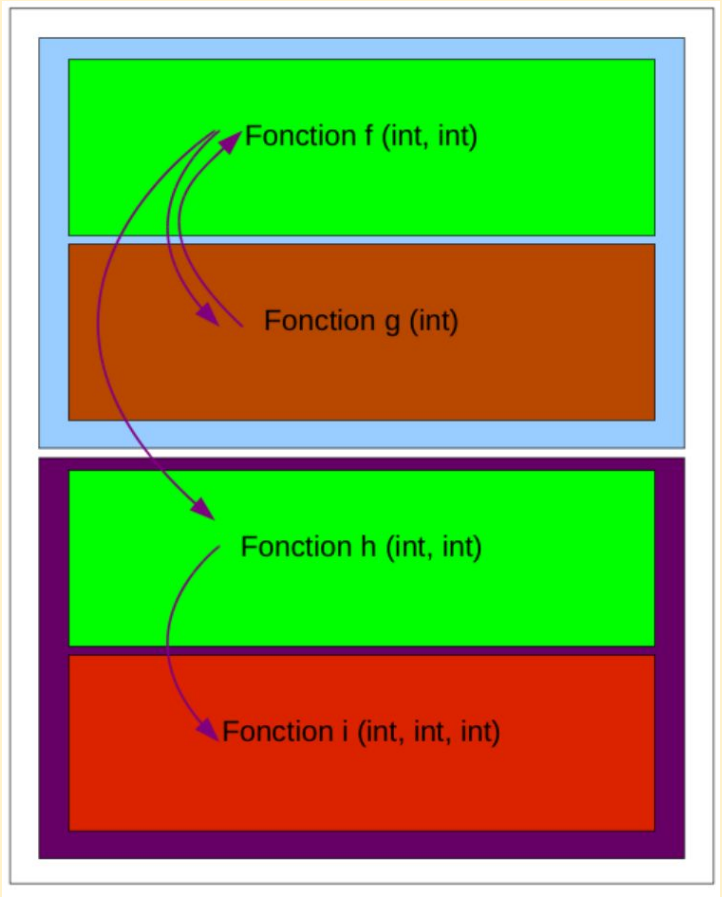
1st order programming



From YRG slides POCA

A ⇒ B means A references B

1st order programming

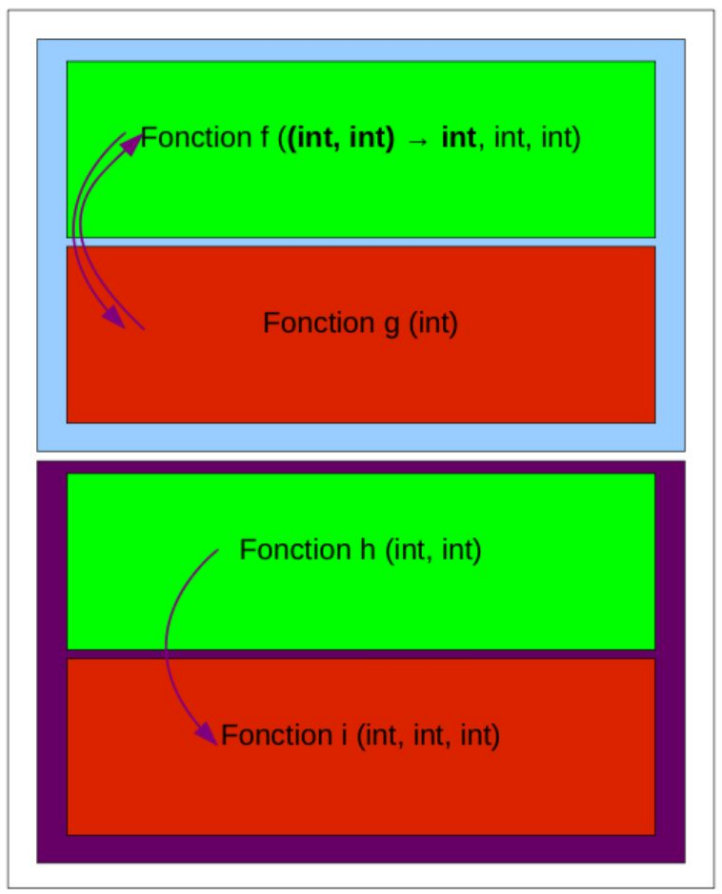


From YRG slides POCA

A ⇒ B means A references B

Higher-order programming

- The definition of a component is dissociated from its use
- The components are independent and thus reusable



From YRG slides POCA

A ⇒ B means A references B

Programming paradigms

- Imperative
- Declarative
- Functional
- Event-driven
- Object-oriented

Programming paradigms: Imperative

Imperative programming focuses on describing how a program operates step by step, rather than on high-level descriptions of its expected results.

Programming paradigms: Imperative

Imperative programming focuses on describing how a program operates step by step, rather than on high-level descriptions of its expected results.

```
1   int myArray[] = {25, 50, 75, 100};
2   int i;
3
4   for (i = 0; i < 4; i++) {
5     printf("%d\n", myArray[i]);
6   }
```

Programming paradigms: Declarative

declarative programming—a style of building the structure and elements of computer programs—that expresses the logic of a computation without describing its control flow.

Lloyd, J.W., *Practical Advantages of Declarative Programming*

Programming paradigms: Declarative

declarative programming—a style of building the structure and elements of computer programs—that expresses the logic of a computation without describing its control flow.

```
cat(tom).                          % tom is a cat
mouse(jerry).                      % jerry is a mouse

animal(X) :- cat(X).               % each cat is an animal
animal(X) :- mouse(X).             % each mouse is an animal

big(X)   :- cat(X).                % each cat is big
small(X) :- mouse(X).              % each mouse is small

eat(X,Y) :- mouse(X), cheese(Y). % each mouse eats each cheese
eat(X,Y) :- big(X),   small(Y).  % each big being eats each small being
```

PROLOG (PROgramming in LOGic)

Programming paradigms: Declarative

declarative programming—a style of building the structure and elements of computer programs—that expresses the logic of a computation without describing its control flow.

```sql
SELECT nom, service
FROM   employe
WHERE  statut = 'stagiaire'
ORDER  BY nom;
```

SQL (Structured Query Language)

Programming paradigms: Functional

Functional programming is a programming paradigm where programs are constructed by applying and composing functions. It is a declarative programming paradigm in which function definitions are *trees of expressions that map values to other values*, rather than a sequence of imperative statements which update the running state of the program.

Programming paradigms: Functional

Functional programming is a programming paradigm where programs are constructed by applying and composing functions. It is a declarative programming paradigm in which function definitions are *trees of expressions that map values to other values*, rather than a sequence of imperative statements which update the running state of the program.

Imperative style

```
const numList = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
let result = 0;
for (let i = 0; i < numList.length; i++) {
  if (numList[i] % 2 === 0) {
    result += numList[i] * 10;
  }
}
```

Functional style

```
const result = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
               .filter(n => n % 2 === 0)
               .map(a => a * 10)
               .reduce((a, b) => a + b, 0);
```

OCaml

Programming paradigms: Functional

Functional programming is a style of programming where programs are made of a composition of *PURE* functions acting on *IMMUTABLE* data.

- Pure functions: always compute the same outputs given the same inputs. The evaluation of a pure function do not produce observable side-effects, e.g., altering the value of a global/shared variable
- Immutable data:

```python
a = [1, 2, 3, 4, 5]
a.insert(0, 0)
print(a)
```

Python

```
let a = [1; 2; 3; 4; 5];;
0::a;;
a;;
```

OCaml

Programming paradigms: Object-Oriented

Object-oriented programming (OOP) is a programming paradigm based on the concept of objects, which can contain data and code

Programming paradigms: Object-Oriented

Object-oriented programming (OOP) is a programming paradigm based on the concept of objects, which can contain data and code

```python
class Account:
    def __init__(self, id, currency, balance):
        self.id = id
        self.currency = currency
        self.balance = balance

    def __str__(self):
        return f"Account ID: {self.id}, Currency: {self.currency}, Balance: {self.balance}"

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        if self.balance >= amount:
            self.balance -= amount
        else:
            print("Insufficient balance")

    def balance(self):
        return self.balance
```

Python

What is Object-Oriented Programming?

- Objects
  - state
  - behavior
  - identity
- Relations between objects
  - message
  - interface
  - abstraction
- Class/instance
- Inheritance
- Polymorphism

From Franck Ledoux's slides *Génie Logiciel - UML*

Why Object-Oriented?: Advantages of OOP

- Stability of the model with regard to the real-world entities
- Adequation with iterative development cycle
- Equilibrium between processing and data
- Possibility to reuse/bring elements from other development
- Simplicity of the model:
  - Objects, messages, classes, inheritance and polymorphism
- Develop software based on:
  - mirroring real-world objects
  - the use of a model independent from the implementation language
- Better understanding of the needs
- Cleaner conception
- System more simple to maintain

From Franck Ledoux's slides *Génie Logiciel - UML*

OOP in Python … Everything is an object

OOP in Python … Everything is an object

```
1   a = ['toto', 5, 2e-1]
2
3   for x in a:
4       print(type(x))
```

```
<class 'str'>
<class 'int'>
<class 'float'>
```

```
11  c = 1. .__add__(3)
12  print(c)
```

```
7   b = (5).__add__(3)
8
9   print(b)
```

A minimal class in Python

```
class Account:
    pass
```

Attributes

```
class Account:
    def __init__(self, id, currency, balance):
        self.id = id
        self.currency = currency
        self.balance = balance
```

```
acc1 = Account(1, "USD", 1000)

acc1.__dict__
```

Methods

```
class Account:
    def __init__(self, id, currency, balance):
        self.id = id
        self.currency = currency
        self.balance = balance

    def __str__(self):
        return f"Account ID: {self.id}, Currency: {self.currency}, Balance: {self.balance}"

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        if self.balance >= amount:
            self.balance -= amount
        else:
            print("Insufficient balance")

    def balance(self):
        return self.balance
```