01_Fundamentals

January 18, 2025

1 Modules/packages/libraries

Definitions:

- Modules: A module is a file which contains python functions, global variables etc. It is nothing but .py file which has python executable code / statement.
- Packages: A package is a namespace which contains multiple package/modules. It is a directory which contains a special file `__init__.py`
- Libraries: A library is a collection of various packages. There is no difference between package and python library conceptually.

Modules/packages/libraries can be easily "imported" and made functional in your python code. A set of libriaries comes with every python installation. Others can be installed locally and then imported. Your own code sitting somewhere else in your local computer can be imported too.

Further details (very important!) on packages and how to create them can be found online. We may find the need of creating our own during the course.

1.1 notes from lecture

Jupyter stands from Ju-lia, Py-thon and R. these are the three programming languages that Jupyter was made for.

Modules is what i built up to this day.

cross-compatibility of libraries was a real issue with c++ libraries. In python with pip everything is checked automatically and you do not have to know and care about these things.

every know and then there might be an update to some library that breaks up cross-compatibility with other libraries. gith management of those libraries is a complicated thing.

[3]:
```
###### all the "stuff" that is in the math library can be used
import math
print(math.pi)

# you can give math a label for convenience
import math as m
print (m.pi)

# alternatively you can import only a given "thing" from the library
```

```
from math import pi    #you can add several libraries at once, just list them␣
 ↪separated by a ", "
print (pi)

# or just get everything (very dangerous!!!)
"""
dangerous because you do not know what is inside the library
never do this unless you know very well what is inside
for example, you can do it when you import a module that you wrote yourself
in math there may be variables and then you declare them again and you are␣
 ↪losing them
and you make a mess
"""
#from math import *
#print (sqrt(7))
```

```
3.141592653589793
3.141592653589793
3.141592653589793
2.6457513110645907
```

To know which modules are there for you to use just type:

[2]:
```
print (help('modules') )
```

```
Please wait a moment while I gather a list of all available modules…

test_sqlite3: testing with SQLite version 3.42.0

2024-10-23 10:53:10.389 Python[1889:51019] WARNING: Secure coding is not enabled
for restorable state! Enable secure coding by implementing
NSApplicationDelegate.applicationSupportsSecureRestorableState: and returning
YES.

IPython             _tracemalloc        heapq               reprlib
PIL                 _typing             hmac                resource
__future__          _uuid               html                rlcompleter
__hello__           _warnings           http                runpy
__phello__          _weakref            idlelib             sched
_abc                _weakrefset         imaplib             scipy
_aix_support        _xxinterpchannels   imghdr              seaborn
_ast                _xxsubinterpreters  importlib           secrets
_asyncio            _xxtestfuzz         inspect             select
_bisect             _zoneinfo           io                  selectors
_blake2             abc                 ipaddress           shelve
_bz2                aifc                ipykernel           shlex
_codecs             antigravity         ipykernel_launcher  shutil
_codecs_cn          appnope             itertools           signal
_codecs_hk          argparse            jedi                site
```

```
_codecs_iso2022     array               json                six
_codecs_jp          ast                 jupyter             smtplib
_codecs_kr          asttokens           jupyter_client      sndhdr
_codecs_tw          asyncio             jupyter_core        socket
_collections        atexit              keyword             socketserver
_collections_abc    audioop             kiwisolver          sqlite3
_compat_pickle      base64              lib2to3             sre_compile
_compression        bdb                 linecache           sre_constants
_contextvars        binascii            locale              sre_parse
_crypt              bisect              logging             ssl
_csv                builtins            lzma                stack_data
_ctypes             bz2                 mailbox             stat
_ctypes_test        cProfile            mailcap             statistics
_curses             calendar            marshal             string
_curses_panel       cgi                 math                stringprep
_datetime           cgitb               matplotlib          struct
_dbm                chunk               matplotlib_inline   subprocess
_decimal            cmath               mimetypes           sunau
_elementtree        cmd                 mmap                symtable
_functools          code                modulefinder        sys
_hashlib            codecs              multiprocessing     sysconfig
_heapq              codeop              nest_asyncio        syslog
_imp                collections         netrc               tabnanny
_io                 colorsys            nis                 tarfile
_json               comm                nntplib             telnetlib
_locale             compileall          ntpath              tempfile
_lsprof             concurrent          nturl2path          termios
_lzma               configparser        numbers             test
_markupbase         contextlib          numpy               textwrap
_md5                contextvars         opcode              this
_multibytecodec     contourpy           operator            threading
_multiprocessing    copy                optparse            time
_opcode             copyreg             os                  timeit
_operator           crypt               packaging           tkinter
_osx_support        csv                 pandas              token
_pickle             ctypes              parso               tokenize
_posixshmem         curses              pathlib             tomllib
_posixsubprocess    cycler              pdb                 tornado
_py_abc             dataclasses         pexpect             tqdm
_pydatetime         datetime            pickle              trace
_pydecimal          dateutil            pickletools         traceback
_pyio               dbm                 pip                 tracemalloc
_pylong             debugpy             pipes               traitlets
_queue              decimal             pkgutil             tty
_random             decorator           platform            turtle
_scproxy            difflib             platformdirs        turtledemo
_sha1               dis                 plistlib            types
_sha2               doctest             poplib              typing
```

```
_sha3               email              posix              tzdata
_signal             encodings          posixpath          unicodedata
_sitebuiltins       ensurepip          pprint             unittest
_socket             enum               profile            urllib
_sqlite3            errno              prompt_toolkit     uu
_sre                executing          pstats             uuid
_ssl                faulthandler       psutil             venv
_stat               fcntl              pty                warnings
_statistics         filecmp            ptyprocess         wave
_string             fileinput          pure_eval          wcwidth
_strptime           fnmatch            pwd                weakref
_struct             fontTools          py_compile         webbrowser
_symtable           fractions          pyclbr             wsgiref
_sysconfigdata__darwin_darwin ftplib             pydoc              xdrlib
_testbuffer         functools          pydoc_data         xml
_testcapi           gc                 pyexpat            xmlrpc
_testclinic         genericpath        pygments           xxlimited
_testimportmultiple getopt             pylab              xxlimited_35
_testinternalcapi   getpass            pyparsing          xxsubtype
_testmultiphase     gettext            pytz               zipapp
_testsinglephase    glob               queue              zipfile
_thread             graphlib           quopri             zipimport
_threading_local    grp                random             zlib
_tkinter            gzip               re                 zmq
_tokenize           hashlib            readline           zoneinfo

Enter any module name to get more help.  Or, type "modules spam" to search
for modules whose name or summary contain the string "spam".

None

/Users/miriamzara/LaboratoryOfComputationalPhysics_Y7/myenv/lib/python3.12/site-
packages/tqdm/auto.py:21: TqdmWarning: IProgress not found. Please update
jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

`pip` is a special package. It is used from the command line to install properly (e.g. matching the version of the local packages) new packages. It can also be used from within python to check i.e. the set installed packages and their versions. N.B.: only the installed packages on top of the default ones will be listed

three numbers e.g. scipy==2.13.1

- *if the first number changes, it surely breaks compatibility*
- *the second one can break compatibility*
- *the third is used to mean fixing bugs*

[ ]:
```
"""""
try:
    # this doesn't work anymore
    import pip
    sorted(["%s==%s" % (i.key, i.version) for i in pip._internal.utils.misc.
 ↪get_installed_distributions()])
    print ("within try")
"""""
# use this instead
import pkg_resources
installed_packages = pkg_resources.working_set
installed_packages_list = sorted(["%s==%s" % (i.key, i.version)
    for i in installed_packages])
print(installed_packages_list)
```

2 Copies and Views

Copies:

- A **copy** creates a new object that is a duplicate of the original one.
- Changes made to the copy do not affect the original object, and vice versa.
- There are two types of copies:
  - **Shallow Copy**: Only the top-level object is copied. If the original object contains references to other objects (e.g., a list of lists), the references are copied, not the objects themselves.
  - **Deep Copy**: A complete copy is made, including all **nested** objects. Changes to any part of the deep copy won't affect the original object.

[5]:
```
import copy
original = [1, 2, [3, 4]]
shallow_copy = copy.copy(original)
print("Original:", original)  # Output: [1, 2, [3, 4]]

shallow_copy[0] = 100  # This won't affect the original
shallow_copy[2][0] = 300  # This will affect the original's nested list

print("Original:", original)  # Output: [1, 2, [300, 4]]
print("Shallow Copy:", shallow_copy)  # Output: [100, 2, [300, 4]]
```

```
Original: [1, 2, [3, 4]]
Original: [1, 2, [300, 4]]
Shallow Copy: [100, 2, [300, 4]]
```

Views:

- A view provides a **reference** to the original object without creating a new object.
- Changes made through the view directly affect the original object because the view is simply another way of accessing the original data.

- We will see in the next classes that NumPy arrays and **Pandas DataFrames often deal with views when you slice or manipulate them**.

this because when you have big data (GB or TB size) you really want to avoid duplicating as much as possible because you dont have that much memory

[ ]:
```
import numpy as np
arr = np.array([1, 2, 3, 4])
view = arr[1:3]  # This creates a view, not a copy

view[0] = 100  # Modifies the original array

print("Original array:", arr)  # Output: [  1 100   3   4]
print("View:", view)  # Output: [100   3]
```

3 Functions

keys are valid when they can be HASHED (more or less: translated into a list of symbols)

for example, lists are hashable and as such they can be used as keys for a dictionary, but most of the times of course you use a string as key

[11]:
```
def square(x):
    """Square of x."""
    return x*x

def cube(x):
    """Cube of x."""
    return x*x*x

# create a dictionary of functions
funcs = {
    'square': square,
    'cube': cube,
}

x = 3
print(square(x))
print(cube(x))

for func in sorted(funcs):
    print (func, funcs[func](x))
```

```
9
27
cube 27
square 9
```

3.1 Functions arguments

In Python, whether arguments passed to a function are treated as views or copies depends on the data type of the argument and how it is handled inside the function. Here's how **Python handles** various types of arguments:

Immutable Types (e.g., integers, strings, tuples): * For immutable types like `int`, `str`, and `tuple`, when you pass them as arguments to a function, Python creates a **copy** of the reference to the object, not the object itself. * Since these objects cannot be modified, **any attempt to change them inside the function results in the creation of a new object**, leaving the original object unchanged.

[8]:
```
def modify(x):
    x += 1
    print("Inside function:", x)

a = 10
modify(a)
print("Outside function:", a)
```

```
Inside function: 11
Outside function: 10
```

Mutable Types (e.g., lists, dictionaries, sets): * For mutable types like `list`, `dict`, or `set`, Python passes a reference to the original object. This means that any modification made to the object inside the function **will affect** the original object outside the function, as they both reference the same object (this behaves like a view).

[9]:
```
def modify_list(lst):
    lst.append(4)
    print("Inside function:", lst)

my_list = [1, 2, 3]
modify_list(my_list)
print("Outside function:", my_list)
```

```
Inside function: [1, 2, 3, 4]
Outside function: [1, 2, 3, 4]
```

3.2 Higher order functions

A function that uses another function as an input argument or returns a function is known as a higher-order function (HOF). The most familiar examples are `map` and `filter`.

Also we look at ***`reduce`*** *and* ***`zip`***

3.2.1 map

The map function applies a function to each member of a collection

[12]:
```
x = list(map(square, range(5)))
print (x)

# Note the difference w.r.t python 2. In python 3 map retuns an iterator so you␣
 ↪can do stuff like:
for i in map(square,range(5)): print(i)

# or
[i for i in map(square,range(6))]
```

```
[0, 1, 4, 9, 16]
0
1
4
9
16
```

[12]:
```
[0, 1, 4, 9, 16, 25]
```

3.2.2 filter

The filter function applies a predicate to each member of a collection, **retaining only** those members where the predicate is True

[14]:
```
def is_even(x):
    return x%2 == 0

print (list(filter(is_even, range(5))))
#print(type(range(5)))
```

```
[0, 2, 4]
<class 'range'>
```

Combinations in sequence of HOF are obviously possible

[ ]:
```
list(map(square, filter(is_even, range(5))))
```

3.2.3 reduce

Not that common anymore but good to know because it was a big thing at the beginning of big data computing

The `reduce` function reduces a collection using a binary operator to combine items two at a time. More often than not reduce can be substituted with a more efficient for loop. It is worth mentioning it for its key role in big-data applications together with map (the map-reduce paradigm). N.B.: it no loger exist as built-in function in python 3, it is now part of the `functools` library

[15]:
```
from functools import reduce

def my_add(x, y):
```

```
    return x + y

# another implementation of the sum function
reduce(my_add, [1,2,3,4,5]) #output = 15
```

[15]: 15

3.2.4 zip

zip is useful when you need to iterate over matched elements of multiple lists

[21]:
```
xs = [1, 2, 3, 4]
ys = [10, 20, 30, 40]
zs = ['a', 'b', 'c','d']

for x, y, z in zip(xs, ys, zs):
    print (x, y, z)

print(list(zip(xs, ys, zs)))

# No problem when sizes are not homogeneous
# Python simply skips the elements with no pairing
xs = [1, 2, 3]
ys = [10, 20, 30, 40]
zs = ['a', 'b', 'c',] # look there is an empty space and it doesnt matter
print(list(zip(xs, ys, zs)))
```

```
1 10 a
2 20 b
3 30 c
4 40 d
[(1, 10, 'a'), (2, 20, 'b'), (3, 30, 'c'), (4, 40, 'd')]
[(1, 10, 'a'), (2, 20, 'b'), (3, 30, 'c')]
```

3.2.5 Custom HOF

[ ]:
```
def custom_sum(xs, transform):
    """Returns the sum of xs after a user specified transform."""
    return sum(map(transform, xs))

xs = range(5)
print (custom_sum(xs, square))
print (custom_sum(xs, cube))
```

3.2.6 Returning a function

[22]:
```
# this is nice!!


def make_logger(target):
    def logger(data):
        with open(target, 'a') as f:
            f.write(data + '\n')
    return logger

foo_logger = make_logger('foo.txt') #foo.txt will be created if not there␣
 ↪already
foo_logger('Hello')
foo_logger('World')
```

the exclamation mark ! at the beginning of the code cell

tells jupyter that what is next needs to be executed from the command line

cat is in fact a bash command

[24]:
```
! cat 'foo.txt'
```

```
Hello
World
```

3.3 Anonimous functions (lambda)

When using functional style, there is often the need to create specific functions that perform a limited task as input to a HOF such as map or filter. In such cases, these functions are often written as anonymous or lambda functions. The syntax is as follows:

lambda *arguments* : *expression*

If you find it hard to understand what a lambda function is doing, it should probably be rewritten as a regular function.

[31]:
```
suml = lambda x,y: x+y
print(suml(3,4), end= '\n\n')
print((lambda x,y: x+y)(3,4), end= '\n\n')
for i in map(lambda x: x*x, range(5)): print (i)
```

```
7

7

0
1
4
```

```
9
16
```

[32]:
```python
# what does this function do?
# answer:
# first make a list with [1, 4, 9, ... 100]
# then sum all the elements
from functools import reduce
s1 = reduce(lambda x, y: x+y, map(lambda x: x**2, range(1,10)))
print(s1)
```

```
285
```

3.4 Recursive functions

[33]:
```python
# nice !!!

def fib1(n):
    """Fib with recursion."""

    # base case
    if n==0 or n==1:
        return 1
    # recursive case
    else:
        return fib1(n-1) + fib1(n-2)


print ([fib1(i) for i in range(10)])
```

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

[34]:
```python
# In Python, a more efficient version that does not use recursion is

def fib2(n):
    """Fib without recursion."""
    a, b = 0, 1
    for i in range(1, n+1):
        a, b = b, a+b # nice!
    return b

print ([fib2(i) for i in range(10)])
```

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

[35]:
```python
# check indeed the timing:
# timeit runs the loop a high number of times, in order to provide you with an␣
 ↪accurate estimate
%timeit fib1(20) # averaged over 1000 loops
```

```
%timeit fib2(20) # this is a thousand factor faster! averaged over 1,000,000␣
↪loops
```

```
674  s ± 1  s per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
359 ns ± 1.2 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
```

3.5 Iterators

for sake of completeness but we will never see them

Iterators represent streams of values. Because only one value is consumed at a time, they use very little memory. Use of iterators is very helpful for working with data sets too large to fit into RAM.

[ ]:
```
# Iterators can be created from sequences with the built-in function iter()

xs = [1,2,3]
x_iter = iter(xs)

print (next(x_iter))
print (next(x_iter))
print (next(x_iter))
#print (next(x_iter))
```

[ ]:
```
# Most commonly, iterators are used (automatically) within a for loop
# which terminates when it encouters a StopIteration exception

x_iter = iter(xs)
for x in x_iter:
    print (x)
```

3.6 More on comprehensions

[ ]:
```
# A generator expression

print ((x for x in range(10)))

# A list comprehesnnion

print ([x**2 for x in range(10)])

# A set comprehension

print ({x for x in range(10)})

# A dictionary comprehension

print ({x: "x" for x in range(10) if x%2==0})
```

3.7 Useful Modules

nowadays this stuff is already inside high level functions of libraries like pandas or numpy

these native modules are not something we will use, but here just for completeness

You may want to have a look at the content of the following modules for further usage of (HO) functions: - operator - functools - itertools - toolz - funcy

3.8 Decorators

Decorators are a type of HOF that take a function and return a wrapped function that provides additional useful properties.

Examples:

- logging
- profiling
- Just-In-Time (JIT) compilation

. .

- logging: you may want to record things happening while the functions is exectued and store in log file
- profiling: almost same thing

[38]:
```
def my_decorator(func):
    def wrapper():
        print("Something is happening before the function is called.")
        func()
        print("Something is happening after the function is called.")
    return wrapper

def say_whee():
    print("Whee!")

say_whee = my_decorator(say_whee) # say_whee is now the decorated function
```

[39]:
```
say_whee()
```

```
Something is happening before the function is called.
Whee!
Something is happening after the function is called.
```

Python allows you to use decorators in a **simpler way** with the @ symbol, sometimes called the **"pie" syntax**

[40]:
```
def my_decorator(func):
    def wrapper():
        print("Something is happening before the function is called.")
        func()
        print("Something is happening after the function is called.")
```

```
    return wrapper

@my_decorator
def say_whee():
    print("Whee!")
```

[41]:
```
say_whee()
```

```
Something is happening before the function is called.
Whee!
Something is happening after the function is called.
```

JIT A JIT (Just-In-Time) compiler refers to a technique used to **improve the performance** of code execution by **compiling** code into machine code at runtime, rather than interpreting it line by line. This helps speed up program execution by reducing the overhead of repeatedly interpreting code.

While the standard Python interpreter, CPython, does not include a JIT compiler, there are alternative Python implementations that provide JIT compilation, such as PyPy or numba.

JIT combines both interpretation and compilation. It interprets code initially, and as it identifies sections of code that are executed repeatedly, it compiles them to machine code, optimizing performance dynamically.

remember! python is slow because it is not compiled, it is interpreted.

[ ]:
```
from numba import jit

@jit
def fast_sum(arr):
    total = 0
    for i in arr:
        total += i
    return total
```

[ ]:
```
fast_sum(list(range(100)))

# is it really fast? Try to compare with my_add above
```

4 Classes and Objects

classes and objects are obsolete now (since 20 years). we do not need to have our main focus on this anymore.Nowadays our main focus is manipulation of enormous datasets in a distributed environment. this is what unabled AI to ramp up. Object-oriented programming is still here and still used, but nowadays the crucial things are others

Old school object-oriented programming is possible and often used in python. Classes are defined similarly to standard object-oriented languages, with similar functionalities.

The main python doc page is worth reading through

[42]:
```
class Pet:
    # the "constructor"
    def __init__(self, name, age):  #inizialize the elements of the class
        self.name=name
        self.age=age
    # class functions take the "self" parameter !!!
    def set_name(self,name):
        self.name=name
    def convert_age(self,factor):
        self.age*=factor

buddy=Pet("buddy",4)
print (buddy.name, buddy.age)
buddy.age=3
print (buddy.convert_age(4))
print (buddy.age)
```

```
buddy 4
None
12
```

[43]:
```
# ineritance is straightforward


# new thing to know!!

class Dog(Pet):
    # the following variables is "global", i.e. holds for all "Dog" objects
    species = "mammal"
    # functions can be redefined as usual
    def convert_age(self):
        self.age*=7
    def set_species(self, species):
        self.species = species

puppy=Dog("tobia",10)
print(puppy.name)
puppy.convert_age()
print(puppy.age)
```

```
tobia
70
```