Modules

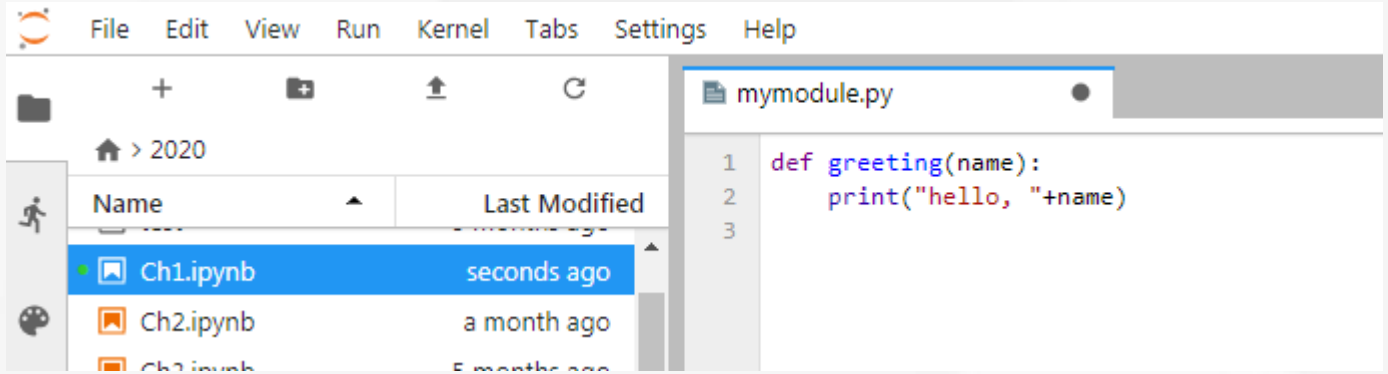
- A module **is a file** containing **functions, classes and variables.** A module can also include **runnable code**;
- The file name is the module name with the *.py* appended.
- You can name the module file whatever you like, but it must have the file extension *.py*
- You can save the module file in the **current directory** of your project.
- The runnable code are executed only the *first* time the module name is encountered in an import statement.

Modules

To create a module just save the code you want in a file with the file extension .py

```
def greeting(name):
    print("hello, "+name)
```

Save this code in a file named mymodule.py

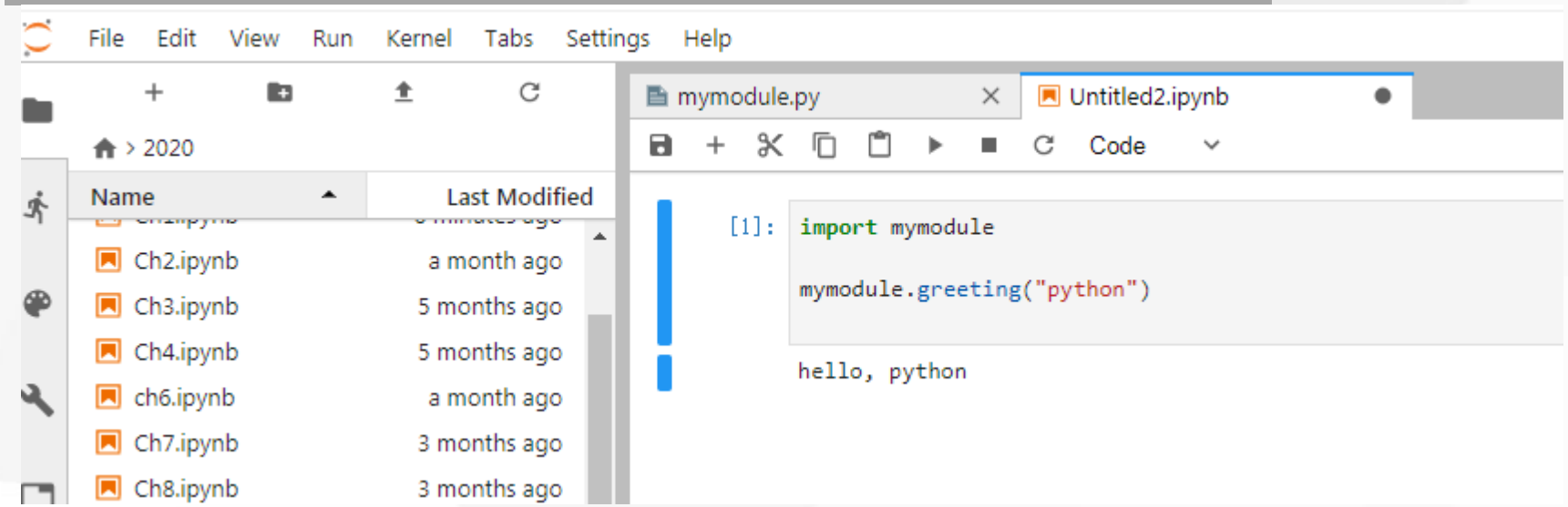


Modules

Import the module named mymodule by the key word **import**, and then call the greeting function by dot (.) operator.

```
import mymodule

mymodule.greeting("python")
```



Modules

Variables in Module: The module can contain functions, but also variables of all types (arrays, dictionaries, objects etc):

```
person1={"name":"John", "age":36,"country":"Norway"}
```

Save this code in a file named mymodule.py

```
import mymodule

a=mymodule.person1["name"]
print(a)
```

Modules

You can choose to import only parts from a module, by using the *from* keyword

```
person1={"name":"Jonh","age":36,"conuntry":"Norway"}

def greeting(name):
    print("hello, "+name)
```

Save this code in a file named mymodule.py

```
from mymodule import person1

print(person1["age"])
```

36

Modules

- When import the module, the *runnable code (NOT function, class, datatype) will be run immediately*
-

```
a=3
print("module,", a)

def greeting(name):
    print("hello, "+name)
```

Save this code in a file named mymodule.py

```
import mymodule

mymodule.greeting("John")
```

module, 3
hello, John

Modules

- Within a module, the module's name (as a string) is available as the value of the global variable ***__name__***.

- If a module is executed directly, the value of the global variable *__name__* will be ***"__main__".***

Modules

```
''' Module fib.py '''

def even_fib(n):
    total = 0
    f1, f2 = 1, 2
    while f1 < n:
        if f1 % 2 == 0:
            total = total + f1
        f1, f2 = f2, f1 + f2
    return total

if __name__=="__main__":
    print(even_fib(1000))
```

- created a ***module*** and saved as **fib.py**.

Modules

You **can run the module directly at the command line**. In this case, the module's __name__ variable has the value "__main__".

```
!python fib.py
```

```
output:4613732
```

```
''' Module fib.py '''

def even_fib(n):
    total = 0
    f1, f2 = 1, 2
    while f1 < n:
        if f1 % 2 == 0:
            total = total + f1
        f1, f2 = f2, f1 + f2
    return total

if __name__=="__main__":
    print(even_fib(4000000))
```

Modules

You can **import the module into the main program**. In this case, the value of __name__ is simply the name of the module itself.

```
import fib
fib.even_fib(1000000)
```

```
output:1089154
```

Mini module quiz

- Given two modules, foo.py and bar.py.

```
''' Module bar.py '''

print("Hi from bar's top level!" )

def print_hello():
    print("Hello from bar!" )

if __name__ == "__main__":
    print("bar's __name__ is __main__")
```

```
''' Module foo.py'''

import bar

print("Hi from foo's top level!")

if __name__ == "__main__":
    print("foo's __name__ is __main__")
    bar.print_hello()
```

- Try to guess the output for each of the following execution methods.

Mini module quiz

```
''' Module bar.py '''

print("Hi from bar's top level!" )

def print_hello():
    print("Hello from bar!" )

if __name__ == "__main__":
    print("bar's __name__ is __main__")
```

```
''' Module foo.py'''

import bar

print("Hi from foo's top level!")

if __name__ == "__main__":
    print("foo's __name__ is __main__")
    bar.print_hello()
```

```
!python bar.py
Hi from bar's top level!
bar's __name__ is __main__
```

Mini module quiz

```
''' Module bar.py '''

print("Hi from bar's top level!" )

def print_hello():
    print("Hello from bar!" )

if __name__ == "__main__":
    print("bar's __name__ is __main__")
```

```
''' Module foo.py'''

import bar

print("Hi from foo's top level!")

if __name__ == "__main__":
    print("foo's __name__ is __main__")
    bar.print_hello()
```

```
!python foo.py
```

Now what happens when we execute the foo module directly?

```
Hi from bar's top level!
Hi from foo's top level!
foo's __name__ is __main__
Hello from bar!
```

设置

Mini module quiz

请您编辑题干

```
''' Module bar.py '''

print("Hi from bar's top level!" )

def print_hello():
    print("Hello from bar!" )

if __name__ == "__main__":
    print("bar's __name__ is __main__")
```

```
''' Module foo.py'''

import bar

print("Hi from foo's top level!")

if __name__ == "__main__":
    print("foo's __name__ is __main__")
    bar.print_hello()
```

```
import foo
import bar
```

Now what happens when we import the foo module into the interpreter?

作答