Functions and Modular Programming (in Python)

Kristoffer Nielbo

Center for Humanities Computing AArhus|chcaa.io
aarhus university, denmark

CENTER FOR HUMANITIES
COMPUTING AARHUS

Imperative

```
1 sum = 0
2 for x in my_list:
3    sum += x
4 print(sum)
```

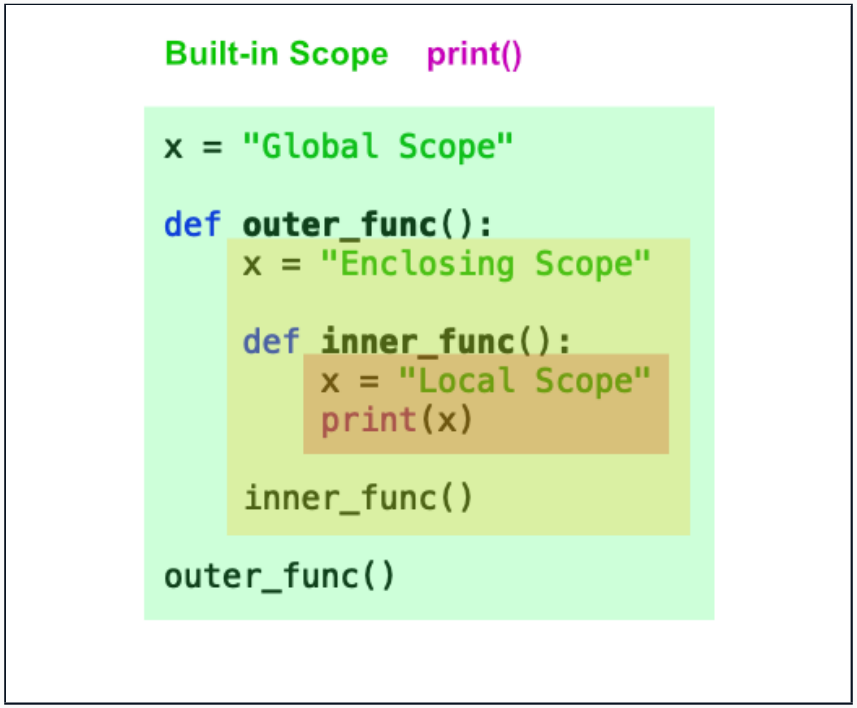
Programming with an explicit sequence of commands that update state.

Procedural

```
1 def do_add(any_list):
2     sum = 0
3     for x in any_list:
4         sum += x
5     return sum
6
7 print(do_add(my_list))
```

Imperative programming with procedure calls.

Variable scope

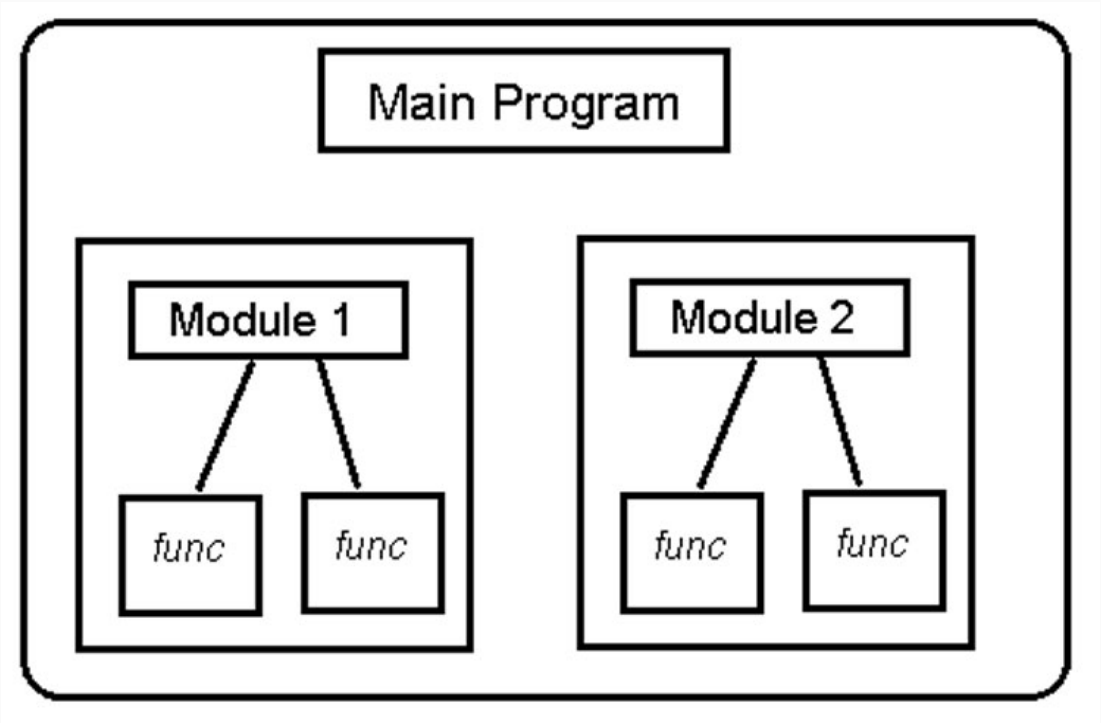


Scope: regions of a program that define the visibility of variables

Global variables reduces the modularity and flexibility of the program! You avoid global variable by passing variables to function arguments.

Modularity

Modularization reduces software complexity and facilitates re-usability



Modular programming breaks the code into parts that can be shared across projects and modified independently

CT Example

Problem `12A4BA78AB11A1314AB`

Decomposition:

- Two types of data, numbers and letters
- Numbers (`int`) are in ascending order
- Letters only `A`, `B`, `AB`
- `AB` only occurs in the end

Pattern recognition:

- `A` in 3^{rd} positions
- `B` in 5^{th} positions
- Numbers are positional values (index)

CT Example

Validate:

x_i	1	2	A	4	B	A	7	8	A	B	11	A	13	14	AB
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

```
1  def solution(n):
2     result = str()
3     for i in range(1, n + 1, 1):
4        flag = False
5        if i % 3 == 0:
6           result += 'A'
7           flag = True
8        if i % 5 == 0:
9           result += 'B'
10          flag = True
11       if flag == False:
12          result += str(i)
13    return result
```

```
1 if questions:
2     try:
3         answer()
4     except RunTimeError:
5         pass
6     else:
7         print('break')
```