

# Scopes

Thomas Schwarz, SJ

# Local and Global Variables

- A Python function is an independent part of a program
  - It has its own set of variables
    - Called local variables
  - It can also access variables of the environment in which the function is called.
    - These are global variables
- The space where variables live is called their scope

# Examples

```
a=3
b=2
def foo(x):
    return a+x
def bar(x):
    b=1
    return b+x

print(foo(3), bar(3))
```

- *a* and *b* are two global variables
- In function *foo*:
  - *a* is global, its value remains 3
- In function *bar*:
  - *b* is local, since it is redefined to be 1

# Examples

- Calculating  $1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$
- Known as the  $n^{\text{th}}$  **harmonic number**

# Examples

- Solution:

```
def harmonic(n):  
    suma = 0  
    for i in range(1,n+1): #1, 2, 3, ..., n  
        suma += 1/i  
    return suma
```

# Examples

- Two-dimensional harmonic
  - A slightly contrived example

$$h_{n,m} = \begin{aligned} & \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{n} \\ & + \\ & \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \dots + \frac{1}{n+1} \\ & + \\ & \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \dots + \frac{1}{n+2} \\ & + \\ & \vdots \\ & + \\ & \frac{1}{m} + \frac{1}{m+1} + \frac{1}{m+2} + \frac{1}{m+3} + \dots + \frac{1}{m+n-1} \end{aligned}$$

# Examples

- We use functions to make our code more understandable
  - Write a helper function that calculates each row
    - Or each column, because the construct is symmetric

- Helper function calculates

$$\frac{1}{r} + \frac{1}{r+1} + \frac{1}{r+2} + \dots + \frac{1}{r+n-1}$$

# Examples

- For a row:
  - $n$  addends, starting with 1 over 1, 2, 3, ...,  $m$
  - Call this  $1+\text{addend}$

```
def gen_harmonic(n, addend):  
    suma = 0  
    for i in range(n):  
        suma += 1/(i+addend)  
    return suma
```



# Examples

- Then we just add up rows:

```
def double_harmonic(n,m):  
    suma = 0  
    for i in range(1, n+1):  
        suma += gen_harmonic(n, i)  
    return suma
```

# Examples

- We have two functions, both use variable suma
  - But despite sharing the same name and being active at the same time
    - They are **different**
    - Because they are in different **scopes**
- We can see that this makes things easier:
  - We do not need to remember all variable names that we used

# Examples

```
def gen_harmonic(n, addend):  
    suma = 0  
    for i in range(n):  
        suma += 1/(i+addend)  
    return suma
```

```
def double_harmonic(n,m):  
    suma = 0  
    for i in range(1, n+1):  
        suma += gen_harmonic(n, i)  
    return suma
```

# Preview of Scoping: The global keyword

- But sometimes we do want to share variables

- ```
def func(a):  
    return a + b
```

- Here, `b` is a variable defined outside of `func`

- Because it is not defined in `func`

- But what if we want to change it in `func` ?

- If we just write

```
def func(a):  
    b = 1  
    return a + b
```

# Preview of Scoping: The global keyword

- Then we create a “local” variable `b`

```
b = 3
```

This `b` is different

```
def func(a):  
    b = 1  
    return a + b
```

From this one

- Thus, the “global” `b` has not changed

# Preview of Scoping: The global keyword

- If we want to change a global variable in a function
  - We use the keyword *global*

```
b = 3
```

```
def func(a):  
    b = 1  
    return a + b
```

```
b = 3
```

```
def func(a):  
    global b  
    b = 1  
    return a + b
```

A global and a local b

Only one b

# Preview of Scoping: The global keyword

- Here is an only slightly artificial example
  - We want to count how often a function is called

**FAILS!**

```
count_called = 0

def func():
    count_called += 1
    return

func()
func()
func()
print(count_called)
```

```
Traceback (most recent call last):
  File "/Users/thomasschwarz/Documents/My
website/Classes/BPMumbai2022/Modules/Lists/
scope_examples.py", line 25, in <module>
    func()
  File "/Users/thomasschwarz/Documents/My
website/Classes/BPMumbai2022/Modules/Lists/
scope_examples.py", line 22, in func
    count_called += 1
UnboundLocalError: local variable
'count_called' referenced before assignment
```

# Preview of Scoping: The global keyword

- Here is an only slightly artificial example
  - We want to count how often a function is called

```
count_called = 0

def func():
    global count_called
    count_called += 1
    return

func()
func()
func()
print(count_called)
```

**Works!**

```
= RESTART: /Users/thomasschwarz/Documents/My
website/Classes/BPMumbai2022/Modules/Lists/
scope_examples.py
3
```



# Example

```
a = 1
b = 2
```

```
def foo():
    global a
    a = 2
    b = 3
    print("In foo:" , "a=", a, " b=", b)
```

```
print("Outside foo: " , "a=", a, " b=", b)
foo()
print("Outside foo: " , "a=", a, " b=", b)
```

```
##Outside foo:  a= 1  b= 2
##In foo: a= 2  b= 3
##Outside foo:  a= 2  b= 2
```

- In foo:
  - A local variable *b*
  - A global variable *a*
  - The value of *a* changes by executing *foo()*

# Scoping

- Scoping is definitely an advanced topic
  - The take-home is:
    - **Don't ever, ever use global variables**
    - **Unless you really need to.**
- Under most circumstances, you should pass variables as arguments.
  - **Python Philosophy: Rules are followed by convention, there is no enforcement**
    - Because sometimes you need to make exceptions