

# Python for Language Processing

## Functions

Dr. Jakob Prange

Fakultät für Angewandte Informatik - Universität Augsburg

CL Fall School 24



Credit: This course is based on material developed by  
Annemarie Friedrich, Stefan Thater, Michaela Regneri, and Marc Schulder at Saarland University

- **Imperative:** *First do this, then do this.*

Procedural Programming. Control Structures execute computational steps, state of the program changes as a function of time.

Commands can be grouped into procedures.

## Example

`Celsius_to_Fahrenheit(c)`

- 1 Multiply `c` with 1.8 and save result to `temp`.
- 2 Add 32 to `temp` and return result of this.

## Elements of imperative programs

### Expressions

- Literals (numbers, strings) ✓
- Variables ✓
- Function Calls ⇐

### Statements

- Assignments ✓
- Control Structures: loops, branches ✓

$$n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 3 \cdot 2 \cdot 1$$

```
1 x = 14
2 r = 1
3 while x > 0:
4     r *= x
5     x -= 1
6
7 print("The factorial of 14 is", r)
```

$$n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 3 \cdot 2 \cdot 1$$

```
1 x = 34
2 r = 1
3 for i in range(x):
4     r *= (i+1)
5
6 print("The factorial of", x, "is", r)
```

Functions are “subprograms” that can (and should) be used to divide a larger problem into several smaller problems.

```
1  def factorial(x) :  
2      """Computes the factorial of x"""  
3      r = 1  
4      for i in range(x) :  
5          r *= (i+1)  
6      return r
```

```
1  def name(var_1, ..., var_n):  
2      """A short documentation (optional)"""  
3      <code>  
4      return <something>
```

- **name**: the name of the function (a variable)
- **var\_1, ..., var\_n**: the parameters of the function
- **return <something>**: usually at the end of the function definition, optional

= assignment of function logic to a variable

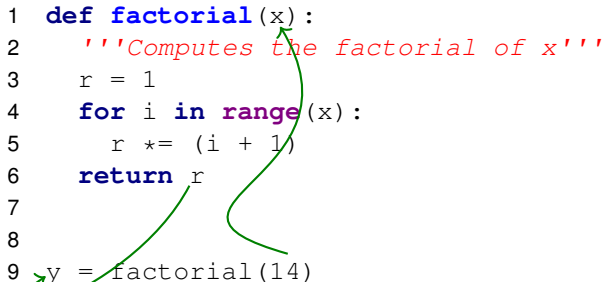
```
1  >>> def factorial(x) :  
2  ...     '''Computes the factorial of x'''  
3  ...     r = 1  
4  ...     for i in range(x) :  
5  ...         r *= (i + 1)  
6  ...     return r  
7  ...  
8  >>> factorial  
9  <function factorial at 0x1e57b0>  
10 >>> help(factorial)  
11 Computes the factorial of x
```



= if a function is applied, the code is executed

```
1  def factorial(x):  
2      '''Computes the factorial of x'''  
3      r = 1  
4      for i in range(x):  
5          r *= (i + 1)  
6      return r  
7  
8  
9  print(factorial(14))    # prints 87178291200  
10 print(factorial(0))    # prints 1
```

```
1 def factorial(x):  
2     '''Computes the factorial of x'''  
3     r = 1  
4     for i in range(x):  
5         r *= (i + 1)  
6     return r  
7  
8  
9 y = factorial(14)
```



A green arrow originates from the argument '14' in the function call 'factorial(14)' on line 9, curves upwards and to the right, and points to the parameter 'x' in the function definition 'def factorial(x):' on line 1. Another green arrow originates from the 'return r' statement on line 6, curves upwards and to the left, and points to the variable 'y' in the assignment 'y = factorial(14)' on line 9.

- When the function is called, the parameters are instantiated with the values from the function call (more specific: *call-by-object-reference*, more on this later!).
- The function call evaluates to the value returned by the function.

```
1 def binary_number(string):  
2     """Returns True if all characters  
3     in string are 0 or 1"""  
4     for c in string:  
5         if c != "0" and c != "1":  
6             return False  
7     return True
```

- The return statement stops the execution of the function and returns a value (or a reference to a value).
- The return statement can occur anywhere in the function definition (not just at the end).