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 Programming Paradigm

- 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.

Imperative Programming Paradigm

Elements of imperative programs

Expressions

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

Statements

- Assignments ✓
- Control Structures: loops, branches ✓

Example: Factorial

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

```python
x = 14
r = 1
while x > 0:
    r *= x
    x -= 1

print("The factorial of 14 is", r)
```

Example: Factorial

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

```python
x = 34
r = 1
for i in range(x):
    r *= (i+1)

print("The factorial of", x, "is", r)
```

Functions

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

```python
def factorial(x):
    """Computes the factorial of x"""
    r = 1
    for i in range(x):
        r *= (i+1)
    return r
```

Anatomy of a Function

```
def name(var_1, ..., var_n):
    """A short documentation (optional)"""
    <code>
    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

Function Definition

= assignment of function logic to a variable

```
>>> def factorial(x):
... '''Computes the factorial of x'''
...   r = 1
...   for i in range(x):
...     r *= (i + 1)
...   return r
...
>>> factorial
<function factorial at 0x1e57b0>
>>> help(factorial)
Computes the factorial of x
```

Function Application

= if a function is applied, the code is executed

```python
def factorial(x):
  '''Computes the factorial of x'''
  r = 1
  for i in range(x):
    r *= (i + 1)
  return r


print(factorial(14))   # prints 87178291200
print(factorial(0))    # prints 1
```

Function Application

```python
def factorial(x):
    '''Computes the factorial of x'''
    r = 1
    for i in range(x):
        r *= (i + 1)
    return r


y = factorial(14)
```

- 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.

The `return` Statement

```python
def binary_number(string):
  """Returns True if all characters
  in string are 0 or 1"""
  for c in string:
    if c != "0" and c != "1":
      return False
  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).