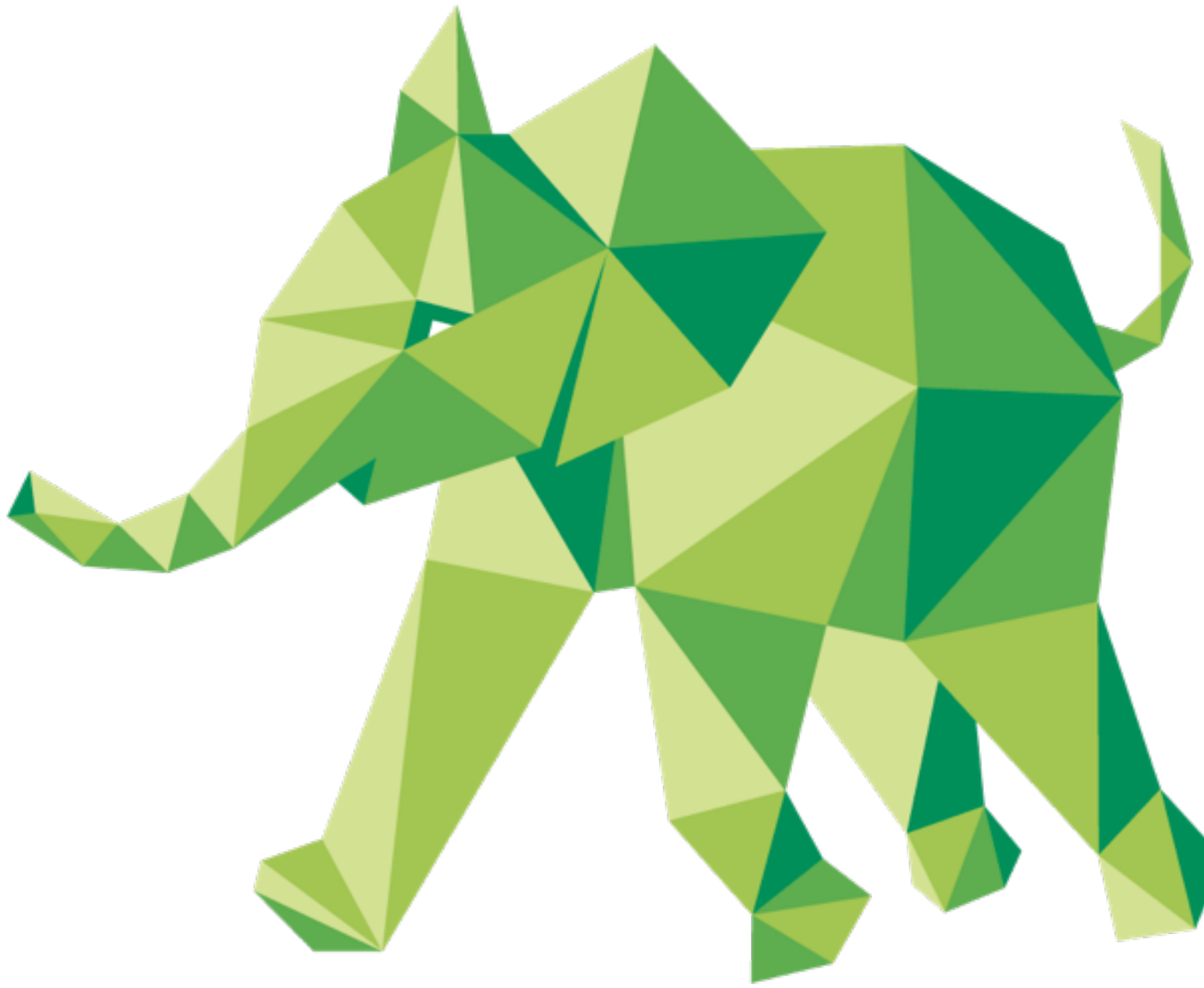
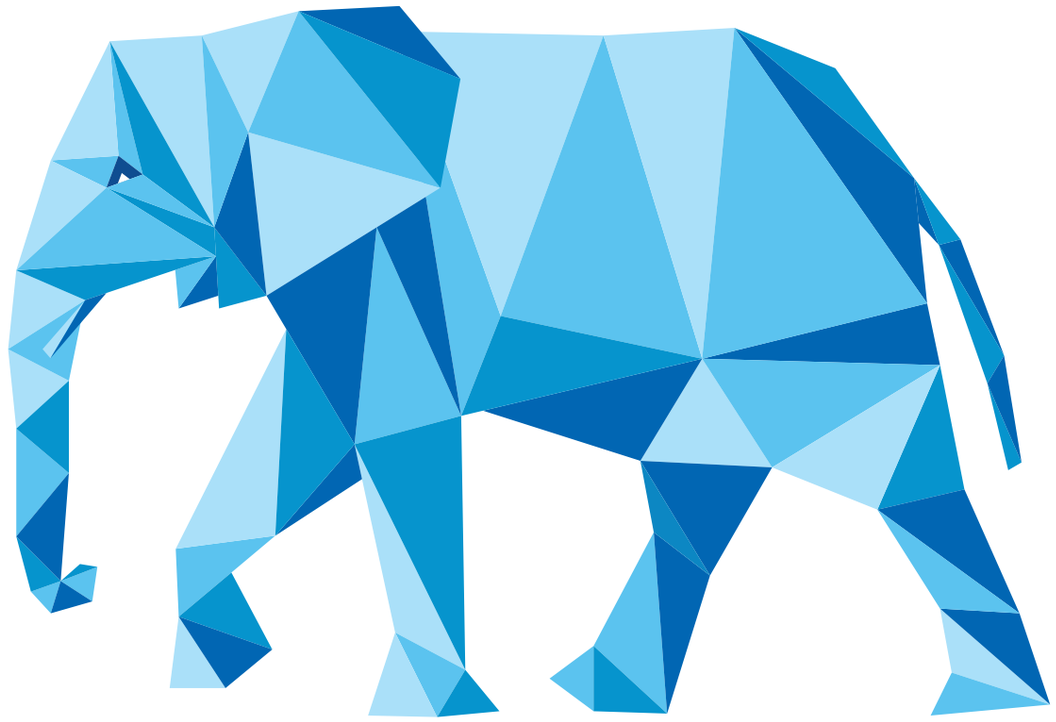


PATWIC

Programming and the world it creates!

Functions

Alva Söder



Functions

- organize blocks of code
- usually take some input and produce some output
- many pre-defined functions in the standard library
- user can define own functions

Calling Functions

- call a function by it's name and all required arguments in parentheses: **len('Hello, world!')**
- the function call is substituted with the **return** value

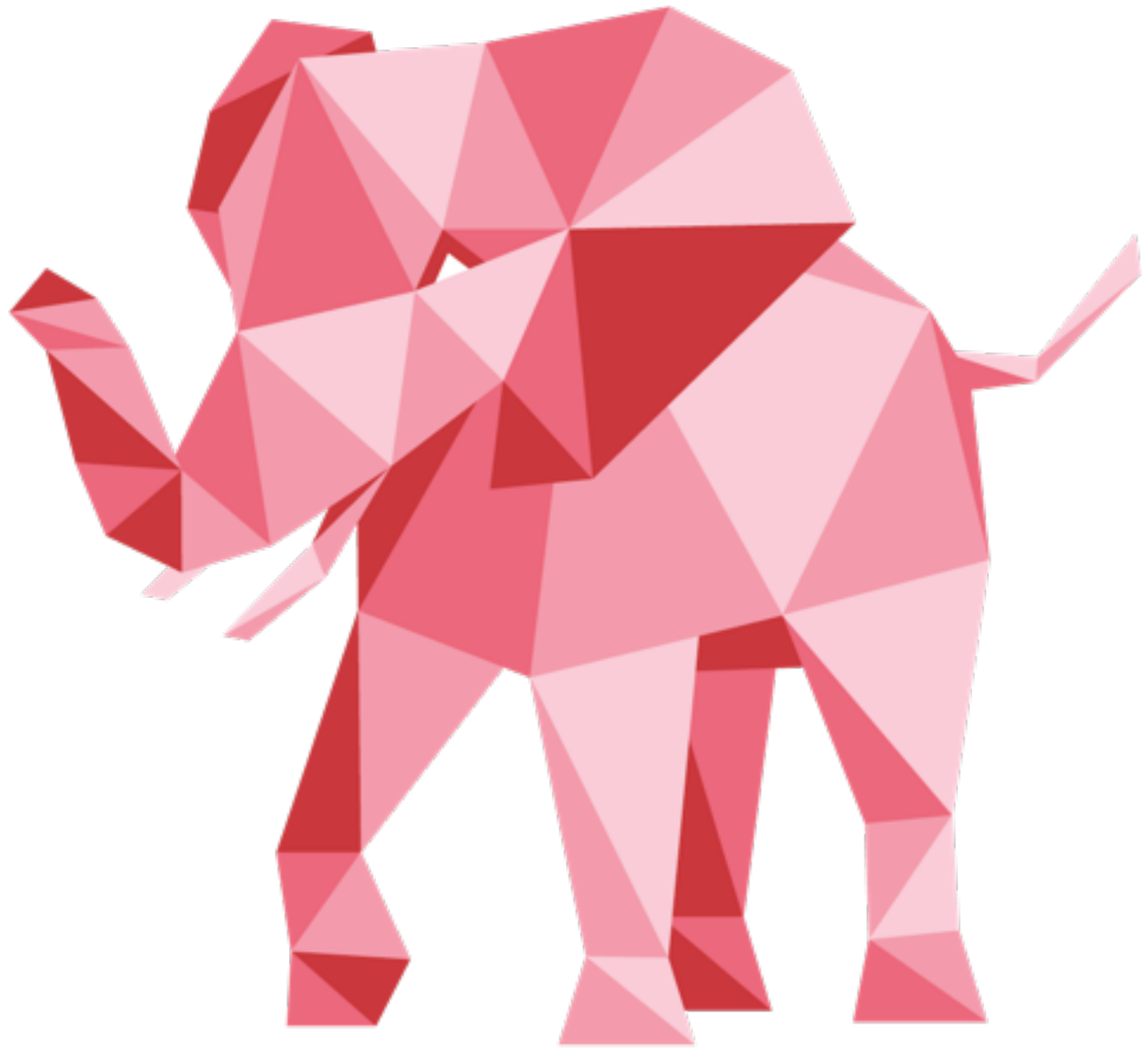
Defining Functions

- the keyword **def**, a **name**, formal **parameters** and an optional **return**

```
def print_multi(n, lo=1, hi=10):

    res = []

    for i in range(lo, hi + 1):
        prod = n * i
        print(str(n) + '*' + str(i) + '=' prod
        res += [prod]

    return res
```

Lists

Nils Reberg



What is a List?

- **ordered sequence** of elements (values of any kind)
- each element resides at a specific **index**
- can be created with square brackets and a list of elements: **cities = ['Lund', 'New York', 'Helsinki']**
- can be accessed with square brackets and a numeric index: **cities[0] = 'Malmö' → ['Malmö', 'New York', 'Helsinki']**
- the empty list is **False**

Common List Operations

- the built-in **len** function returns the number of elements in a list: **len([1, 2, 3]) → 3**

- built-in **sorted** function returns a new list with the same elements sorted according to their natural ordering:
  sorted([2, 3, 1]) → [1, 2, 3]

- can be indexed with a slice in square brackets and the index **[from:to]** from is inclusive, to is exclusive:
  0 1 2 3 4
  [0, 11, 22, 33, 44][1:3] → [11, 22]

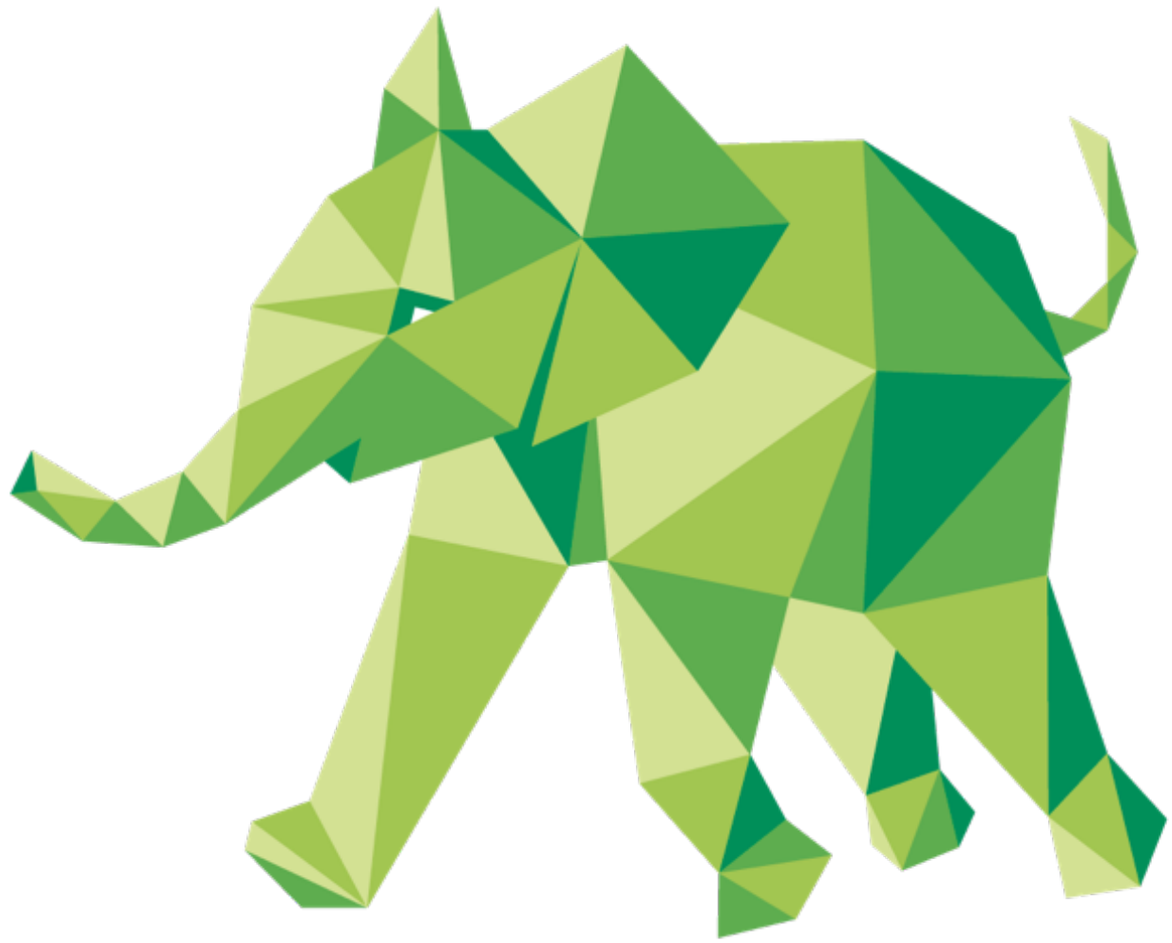
Looping Over Lists

- lists can be used in a **for-loop:** a local variable **n is bound to each element** in the list in turn until the lists has no more elements

```
for n in [1, 2, 3]:
    print(n)
1
2
3

for c in 'Hello, world!':
    print(c.upper())
H
E
L
...
!
```

Useful Tidbits

Alexander Simko



Listing Members

- the **dir** function shows all members of a module or object

- calling without arguments shows names in the **module namespace**

```
>>> dir()
['__builtins__', '__doc__', '__loader__', '__name__', '__package__']
```

- calling with any object as an argument shows **members of that object**

```
>>> s = 'Hello, world!'
>>> dir(s)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__',
'__eq__', '__format__', '__ge__', '__getattribute__',
...
'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith',
'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

Getting Help

- the **help** function shows the documentation
- calling without arguments displays **interactive help**

```
>>> help()
Welcome to Python 3.3!  This is the interactive help utility...

help> math
Help on module math:

NAME
    math...
```

- passing an object as an argument to the help function shows the documentation for that object

```
>>> s = 'Hello, world!'
>>> help(s.upper)
Help on built-in function upper:

upper(...)
    S.upper() -> str

    Return a copy of S converted to uppercase.
(END)
```

Strings

- sequences of characters
- immutable (cannot be changed)
- can be manipulated with methods and built-in functions
- methods return a new string

Common String Operations

- **s.lower()**: returns the string **s** in all lower case
- **s.upper()**: returns the string **s** in all upper case
- **s.split(sep)**: splits a string **s** into a list of words on every occurrence of **sep**
- **sep.join(xs)**: joins the strings in **xs** with **sep** in between
- **find(c)**: returns the index of the first occurrence of **c**

Division

- **division**
  5 / 2 = 2.5

- **integer division**: just the integer part of the result
  5 // 2 = 2

- **modulo**: just the remainder part of the result
  5 % 2 = 1

Logical Operators

- return one of the booleans **True** or **False**
- **in**: does obj exist in some collection
- **and**: combines two boolean expressions and returns True if both expressions are True
- **or**: combines two boolean expressions and returns True if either or both expressions are True
- **not**: negates any boolean expression; True becomes False and vise versa
- **every value is True except 0 and empty strings and collections**

Selection

- **if**, **elif** and **else** select blocks to execute based on the value of boolean expressions

```
if a > b:
      print 'a is larger'
elif a < b:
      print 'a is smaller'
else:
      print 'a and b are equal'
```