Python

Functions

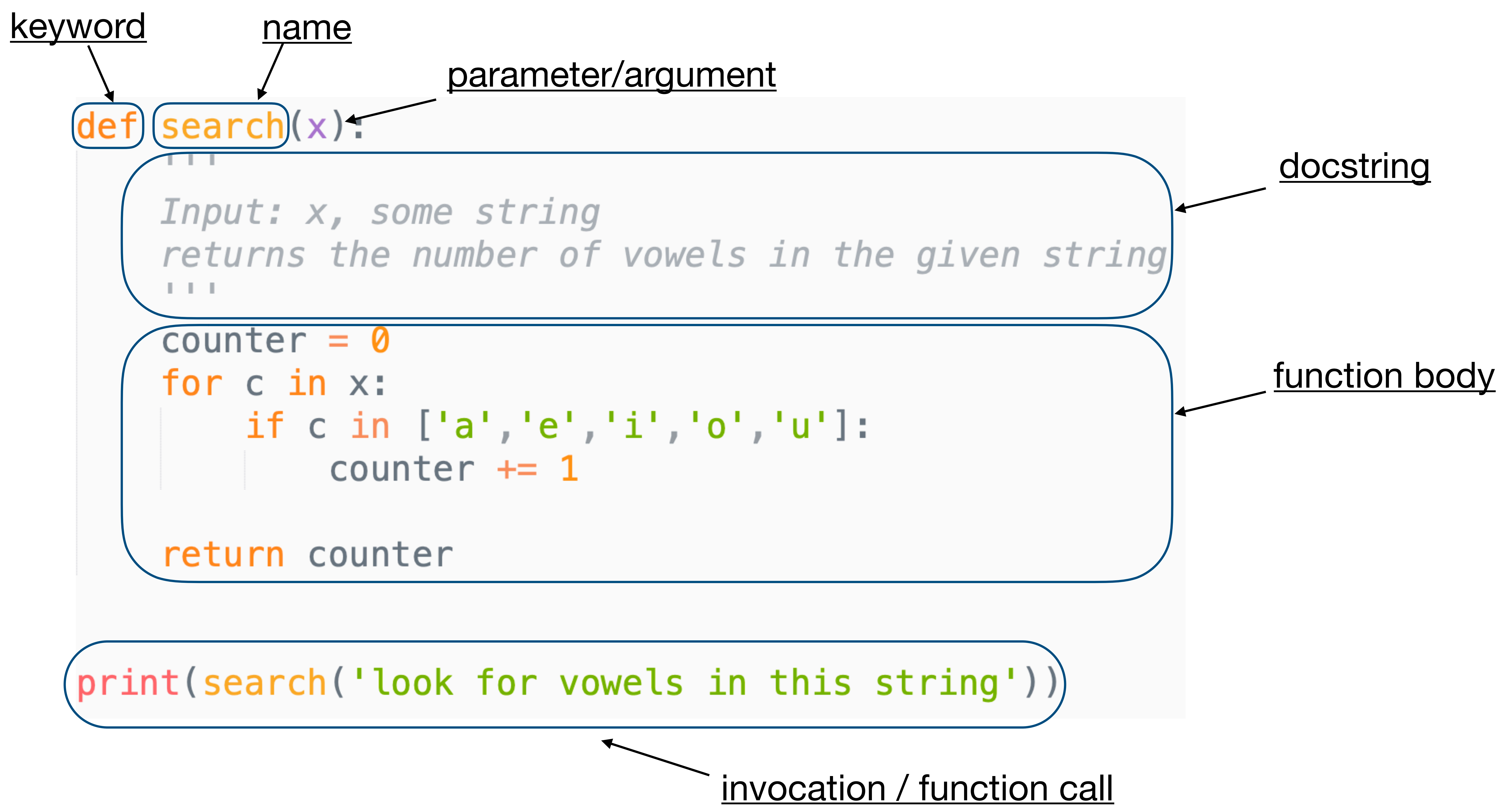
Prof. Dr. Thomas Kopinski

2019

Python | Functions

- anything you want to reuse multiple times should be put into **functions**
- functions are called / invoked, then the code in their body is run
- functions are defined by
  - a name
  - parameters
  - a docstring (optional)
  - a body
  - a return value

Python | Functions



```python
def search(x):
    '''
    Input: x, some string
    returns the number of vowels in the given string
    '''
    counter = 0
    for c in x:
        if c in ['a','e','i','o','u']:
            counter += 1

    return counter


print(search('look for vowels in this string'))
```

Python | return value

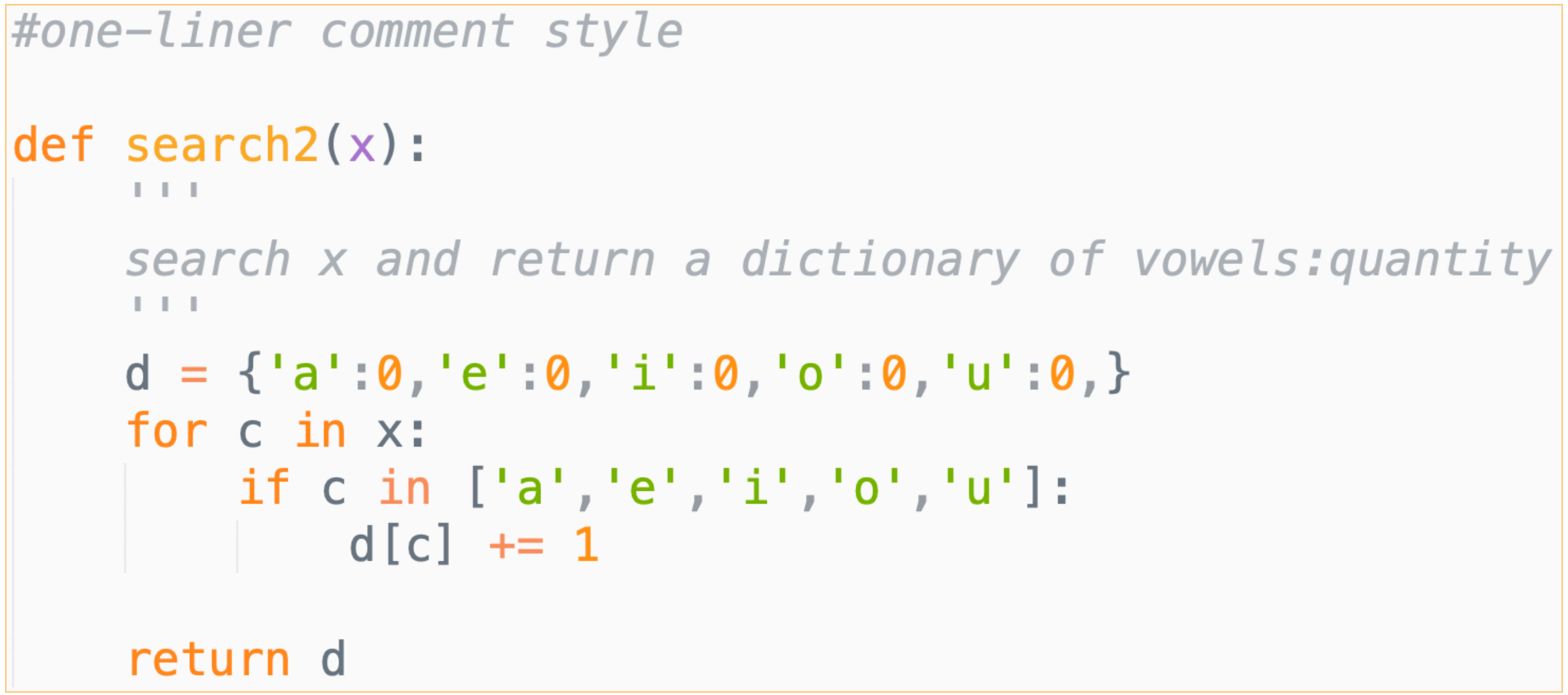
```
def doSomething(x,y):
    c = x+y

res = doSomething(1,2)
print(res)
```

- Python returns None if no return statement is present in a function

Python | Docstrings

- Docstrings are explained in more detail here: https://www.python.org/dev/peps/
- a docstring is a literal at the beginning of a
  - module
  - function
  - class
- it becomes the `__doc__` attribute of that object
- there are various styles (cf. link)



```python
#one-liner comment style

def search2(x):
    '''
    search x and return a dictionary of vowels:quantity
    '''
    d = {'a':0,'e':0,'i':0,'o':0,'u':0,}
    for c in x:
        if c in ['a','e','i','o','u']:
            d[c] += 1

    return d
```

Python | Scopes, lifetime

- variables have a lifetime and a scope
- lifetime: duration, for which the variable exists
- scope: parts of a program, where a variable is accessible
- global vars:
  - are defined in the main program
  - are visible anywhere in the file, where they are defined
  - also visible in all other files which import this file
  - recommendation:
    - use rarely
    - be aware which objects should be global
- local vars:
    - are defined inside a function
    - exist from creation until end of the execution of this function
    - parameters of a function behave like local vars
    - can contain values we pass along directly

Python | Scopes, lifetime

```python
gv = 10

def globTest():
    #print(gv) -- uncommenting this will result in an error
    gv = 20
    print(gv)

globTest()
print (gv)
```

```
20
10
```

```
Traceback (most recent call last):
  File "function.py", line 56, in <module>
    globTest()
  File "function.py", line 52, in globTest
    print(gv)
UnboundLocalError: local variable 'gv' referenced before assignment
```

Python | parameters

- kinds of parameters:
  - positional parameter
  - optional parameter
  - keyword parameter
  - list parameter
  - keyword-only parameter
  - var-keyword parameter

- parameters follow an order and it should be:
  - positional parameter / non-default parameter
  - keyword parameter / default parameter
  - keyword-only parameter
  - var-keyword parameter

```
def exFunc(a, b, c=None, k="xyz" , d=[], *args,  **kwargs):
    return None
```

Note

- empty lists should not be used as they get carried across runs —> use None

Python | Scopes, lifetime

```python
def myFunction(b, s, a=5):
    #b is unused in this example
    print(s)
    for c in s:
        if c == str(a):
            print('found a number')
            break

myFunction(10, 'this is a 5, isnt it')
```

- b, s and a live in the scope of `myFunction`
- non-default arguments should preceded default arguments

Python | **args and **kwargs

```python
def f1(*args):
    for arg in args:
        print(arg)


f1(1, 2, 3)
```

```
1
2
3
```

```python
def f2(**kwargs):
    for k in kwargs.keys():
        print(k + ' : ' + str(kwargs[k]))

args = {'a': 1, 'b': 2, 'c':3}
f2(**args)
```

```
a : 1
b : 2
c : 3
```

Python | *args and **kwargs

- `args, kwargs` | naming convetion —> stick to it!!!
- both `args, kwargs` allow to define a variable number of arguments
- use *`args` to pass along **non-keyworded** arguments
- use *`kwargs` to pass along **keyworded** (named) arguments
- can be used in one function:

```
args = {'a': 1, 'b': 2, 'c':3}
#f2(**args)

def f3(a, b ,c):
    print(a)
    print(b)
    print(c)

f3('this', 2, 'that')
f3(**args)
```

```
this
2
that
1
2
3
```