Utrecht Bioinformatics Center

Functions and Modules

Day 3 – Introduction to Python

UMC Utrecht

Universiteit Utrecht

Hubrecht Institute
Developmental Biology and Stem Cell Research

Utrecht Bioinformatics Center

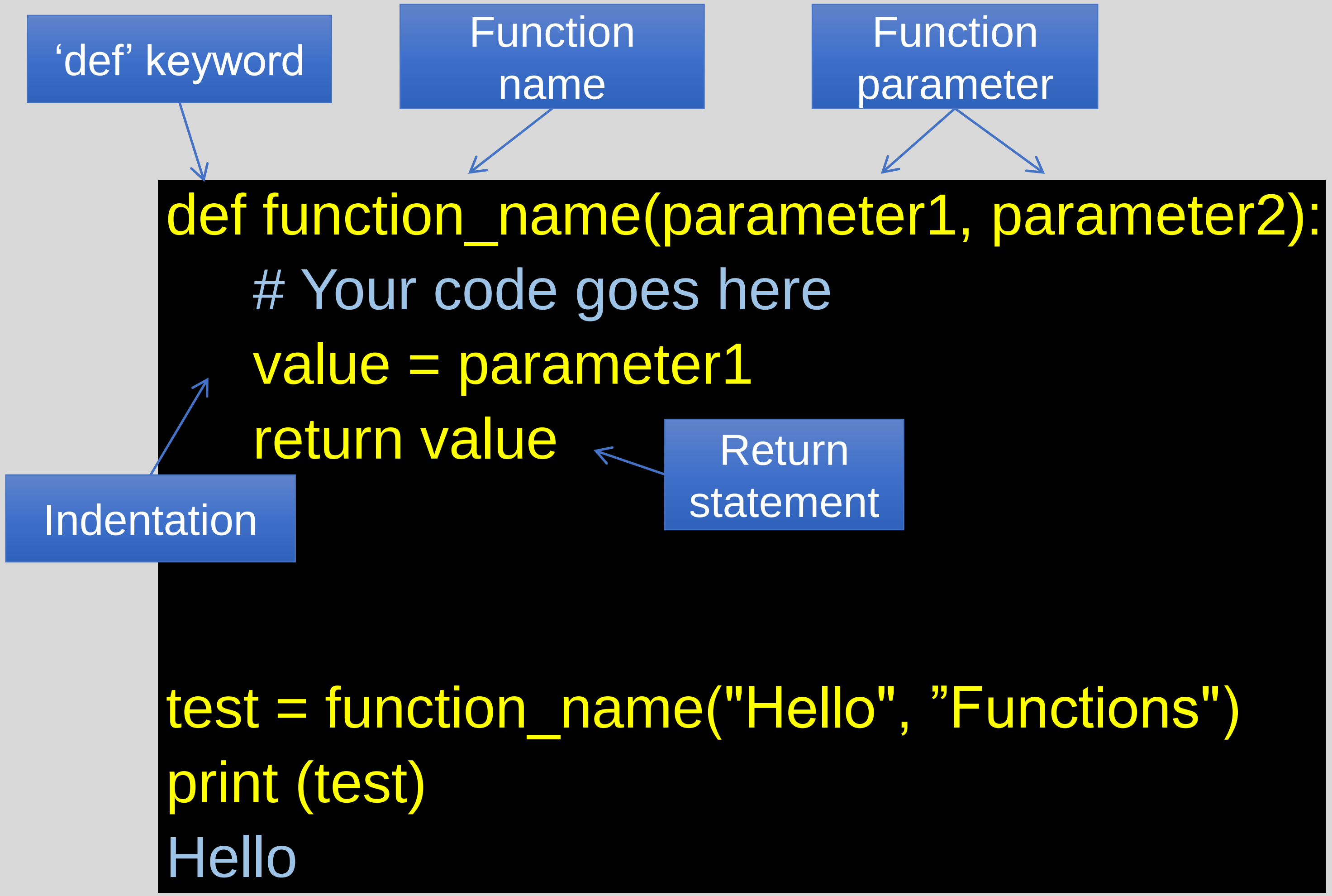
Functions

- Pieces of code to execute a specific task

- Advantages:
  - Reduce duplication of code
  - Complex problems into simpler pieces
  - Improve clarity of the code

Functions

- Pieces of code to execute:



```
def function_name(parameter1, parameter2):
    # Your code goes here
    value = parameter1
    return value


test = function_name("Hello", ”Functions")
print (test)
Hello
```

```
def reverse_it(stuff_i_type):
            text_out = stuff_i_type[::-1]
            return text_out

andersom = reverse_it(“Apple”)
print (andersom)
epplA
```

Functions

- also work without parameters:

```
def hello_functions():
	value = "Hello Functions!"
	return value

hello = hello_functions()
print (hello)
Hello Functions!
```

- can include loops:

```
def print_list(my_list):
	for item in my_list:
	print (item)

test_list = [1,2,3]
print_list(test_list)
1
2
3
```

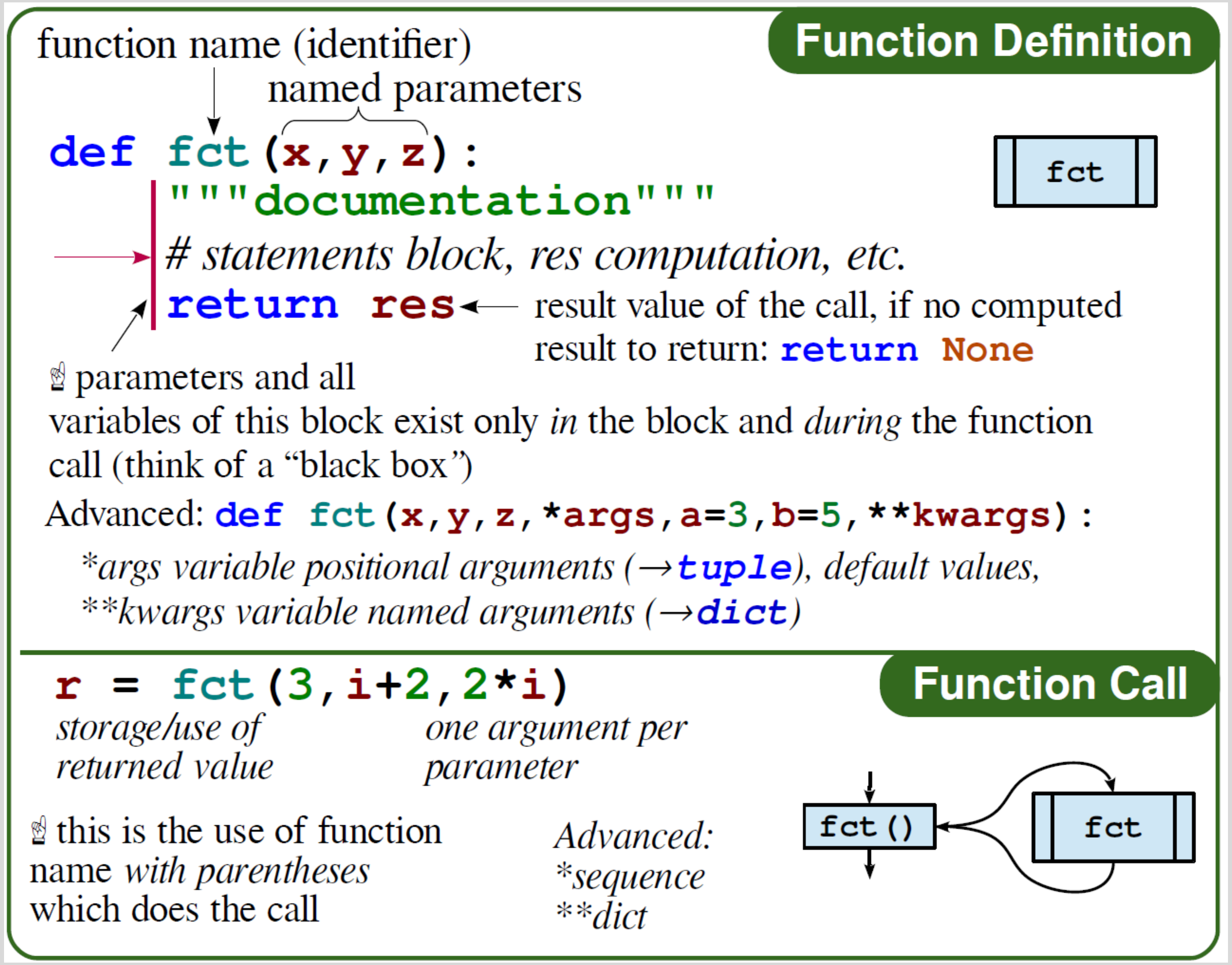
```
def print_even(list_of_numbers):
		for each_number in list_of_numbers
			if(each_number%2.0 == 0):
				print (each_number)

some_numbers = range(10)
print_even(some_numbers)
2
4
6
8
10
```

```
def sum_list(list):  # define function sum_list, with parameter “list”
		total = 0 # make variable “total” with value 0
		for item in list: # loop over the list and
			total += item # add each value of the list to total
		return total # return total

test = [1,2,3]
print (sum_list(test))
6
```

Functions



Function Definition

function name (identifier)

named parameters

```
def fct(x,y,z):
    """documentation"""
    # statements block, res computation, etc.
    return res
```

result value of the call, if no computed result to return: `return None`

☝ parameters and all variables of this block exist only *in* the block and *during* the function call (think of a "black box")

Advanced: `def fct(x,y,z,*args,a=3,b=5,**kwargs):`

**args variable positional arguments (→`tuple`), default values,*
***kwargs variable named arguments (→`dict`)*

Function Call

```
r = fct(3,i+2,2*i)
```

storage/use of returned value

one argument per parameter

☝ this is the use of function name *with parentheses* which does the call

Advanced:
**sequence*
***dict*

Functions and Lists

- Now lets try to write a function to get all codons from the sequence: 'ACGATCGATCGTACGATCGATACG’

```
def get_codons(seq):
    # Make an empty list that will contain codons
    # Loop over seq considering 3 letters at a time
    for codon in range(?,len(seq),?):
            # Append your list with the codon from seq
            # You can return the list containing codons
    return # list with codons


sequence = 'ACGATCGATCGTACGATCGATACG'
print (get_codons(sequence))
```

Functions and Dictionaries

- Write a function to calculate number of times a codon is used in the sequence: 'ACGATCGATCGTACGATCGATACG’.

```
def get_counts(seq):
    # Make an empty dictionary to store codons
    for i in range(0,len(seq),3):
        codon = seq[i:i+3]
        # use the codons as keys and the counts as values
        # If the codon is already within your
        # dictionary then increase its count
        # otherwise add it to the dictionary with count = 1
    return # dictionary with codon counts


sequence = 'ACGATCGATCGTACGATCGATACG'
print (get_counts(sequence))
```

Modules

- Toolbox or kit which is a collection of functions

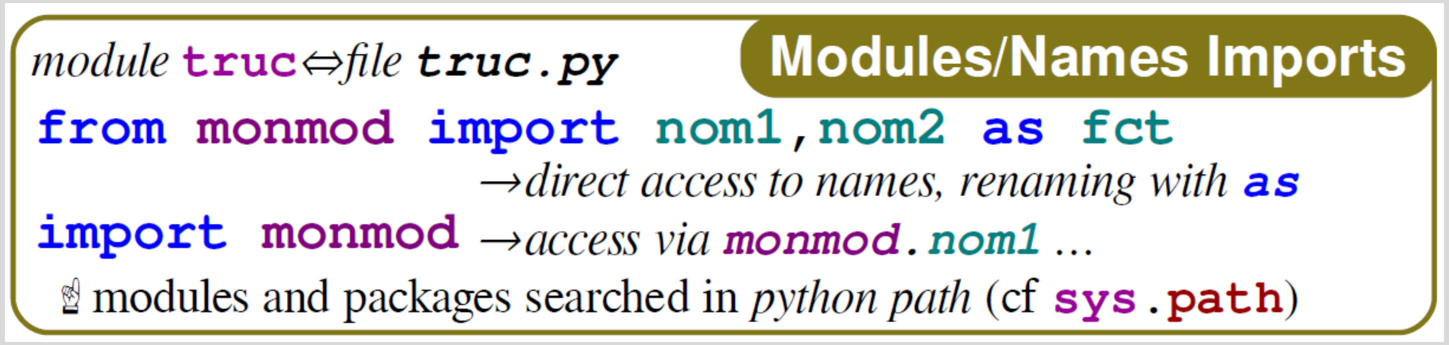
- A file consisting of Python code.

- A module allows you to logically organize your code.
  - To split it into several files for easier maintenance.
  - To reuse that handy function that you’ve written in several programs, without copying it into every script.

- Package is a collection of modules

Built-in Modules

- Some important (default) modules:
  - math
  - itertools
  - random
  - sys
  - dir
  - And many more: https://docs.python.org/3/library/index.html



Modules/Names Imports

module `truc`⇔*file* `truc.py`

```
from monmod import nom1,nom2 as fct
```

→*direct access to names, renaming with* `as`

```
import monmod
```

→*access via* `monmod.nom1` …

☝ modules and packages searched in *python path* (cf `sys.path`)

Modules – making you own

- You will write a function to count the number of times a codon is used.
- You can store such functions in a file so that we can use them later.
- We have stored get_counts and translate_dna into a file called dna_tools.py

- You can import a module in Python using its file name:

```
import dna_tools #This calls the file dna_tools.py stored in the same folder


dna_tools.get_counts('ATCGATCATGAC')
```

Modules – rename

- You can also rename a module when you import it in Python and call the function inside the module.

```
import dna_tools as tools
tools.get_counts('ATCGATCATGAC')
```

- Now try to load your dna_tools module and try both functions

Exercises