

# Functions

**Input**

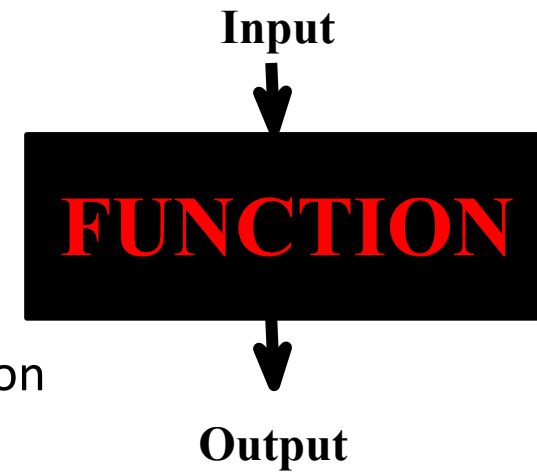


**FUNCTION**



**Output**

# Functions



- Functions are code blocks that only run if called
- Several inputs (parameter) can be passed into a function
- A function can return a result, but does not need to

```
1 def I_am_a_function():  
2 |     print('Hello world!')  
3  
4 I_am_a_function()
```

Hello world!

- Python comes with pre-coded functions!

# Functions

Function indicator      Function name (no spaces in name!)      Argument/parameter

Function is only defined (nothing happens)

```
1 def calculate_sqr_root( number ):
2     """ calculates the sqrt root """
3
4     sqrt_root = number**0.5
5
6     return sqrt_root
7
8 result = calculate_sqr_root(125)
9 print(result)
```

Body of function (everything happens here)

#do something

#return variable (optional)

#function call

Calling function with arguments (definition above required)

11.180339887498949

# Functions

- Number of parameter: If a function expects 2 parameter, the function must be called with 2 parameter; no more, no less
- BUT: Arbitrary arguments (\*args) allow undefined number of arguments



```
1 def I_am_a_function(*inputs):  
2 |   print('Hello world!' + inputs[0] + inputs[1])  
3  
4 I_am_a_function(' Hi!', ' How are you?')
```

Hello world! Hi! How are you?

# Functions

- Default parameters can be used for functions:



```
1 def calculate_sqr_root( number=9 ):
2 |     return number**0.5
3
4 print( calculate_sqr_root() )
5 print( calculate_sqr_root(125) )
```



```
3.0
11.180339887498949
```

# Advantages of functions

- Generate modules => write it ones and apply it often (for different purposes)
- Structure => increases readability of your code
- Nesting of calculations:



```
1 def calc_gc_content( seq ):
2 #     """ calculates GC content """
3 | return float( seq.count('G') + seq.count('C')) / len(seq) *100
4
5 print( round ( calc_gc_content("ATGCGATGCCAAAT"), 2 ), '%' )
```

↳ 42.86 %

- Recursive programming: function can call itself

# Important functions

`str( <VARIABLE> )`                      #converts variable to string

`int( <VARIABLE> )`                      #converts variable to integer

`float( <VARIABLE> )`                      #converts variable to float

`<STRING1>.count( "<STRING2>" )`                      #counts occurrences of string2 in string1

`<LISTE>.count(<LIST_ELEMENT>)`                      #counts occurrences of element in list

`len(<STRING/LISTE>)`                      #calculates length of string/list

- Warning: Functions return error if invalid arguments (e.g. wrong variable type) are given!

# Gap Exercises 2

- Access GitLab exercise folder
- Upload 'gap\_exercise2\_functions.ipynb' into Google Colab
- Fill in/correct the code



# Exercises A – Part2

Primer: “ATGCCATGCATTCGACTACG”

- 2.1) Calculate length of primer and print it!
- 2.2) Get number of Gs and print it!
- 2.3) Write a function to analyze the nucleotide composition of a primer and print it!
- 2.4) Is it a suitable primer? Why (not)?

# Control structures

# if & else

- Distinguish between two cases:  
=> Action depends on result of comparison

```
1 a = 5                                     #define a variable
2 b = int(input('please enter a number: ')) #optional: user input
3
4 if b < a:                                 #if b smaller than a
5     print('b is small than a')           #do something
6 else:                                    #otherwise
7     print('b is NOT smaller than a')     #do something else
```

```
please enter a number: 4
b is small than a
```

# elif

- Distinguish between two cases:  
=> Action depends on result of comparison

```
1 a = 5                                     #define a variable
2 b = int(input('please enter a number: ')) #optional: user input
3
4 if b < a:                                 #if b smaller than a
5 |     print('b is small than a')          #do something
6 elif b==a:                               #if b equal than a
7 |     print('b is equal to a')            #do something
8 else:                                    #otherwise
9 |     print('b is NOT smaller than a')    #do something else
```

please enter a number: 5  
b is equal to a

- if statements can be nested: another if statement inside the body of an if statement

# If & conditions

- Equals:  $a == b$
- Not Equals:  $a != b$
- Less than:  $a < b$
- Less than or equal to:  $a <= b$
- Greater than:  $a > b$
- Greater than or equal to:  $a >= b$

# If & else

- Short writing style:

```
1 a = 5                                #define variables
2 b = 30
3
4 if b < a: print('b is small than a')   #if b smaller than a
5 else: print('b is NOT smaller than a') #otherwise
6 # or
7 print('b is small than a') if b < a else print('b is NOT smaller than a')
```

```
b is NOT smaller than a
b is NOT smaller than a
```

- and & or:

```
1 a, b, c = 5, 30, 60
2
3 if (b > a and b < c) and (a != b or a != c):
4 | print('b larger a, but smaller c AND a not b or a not c')
```

```
b larger a, but smaller c AND a not b or a not c
```

# Gap Exercises 3

- Access GitLab exercise folder
- Upload 'gap\_exercise3\_if\_else.ipynb' into Google Colab
- Fill in/correct the code

# Exercises B – Part1

- 1.1) Write script for guessing numbers!
- 1.2) Add tips (smaller/larger) during the guessing process!