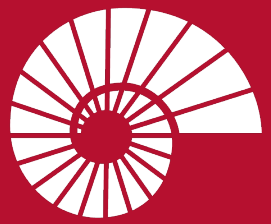


KOÇ
ÜNİVERSİTESİ

06 – Functions and Modules

COMP 125 Programming with Python

Recording Disclaimer

The synchronous sessions are recorded (audiovisual recordings). The students are not required to keep their cameras on during class.

The audiovisual recordings, presentations, readings and any other works offered as the course materials aim to support remote and online learning. They are only for the personal use of the students. Further use of course materials other than the personal and educational purposes as defined in this disclaimer, such as making copies, reproductions, replications, submission and sharing on different platforms including the digital ones or commercial usages are strictly prohibited and illegal.

The persons violating the above-mentioned prohibitions can be subject to the administrative, civil, and criminal sanctions under the Law on Higher Education Nr. 2547, the By-Law on Disciplinary Matters of Higher Education Students, the Law on Intellectual Property Nr. 5846, the Criminal Law Nr. 5237, the Law on Obligations Nr. 6098, and any other relevant legislation.

The academic expressions, views, and discussions in the course materials including the audio-visual recordings fall within the scope of the freedom of science and art.

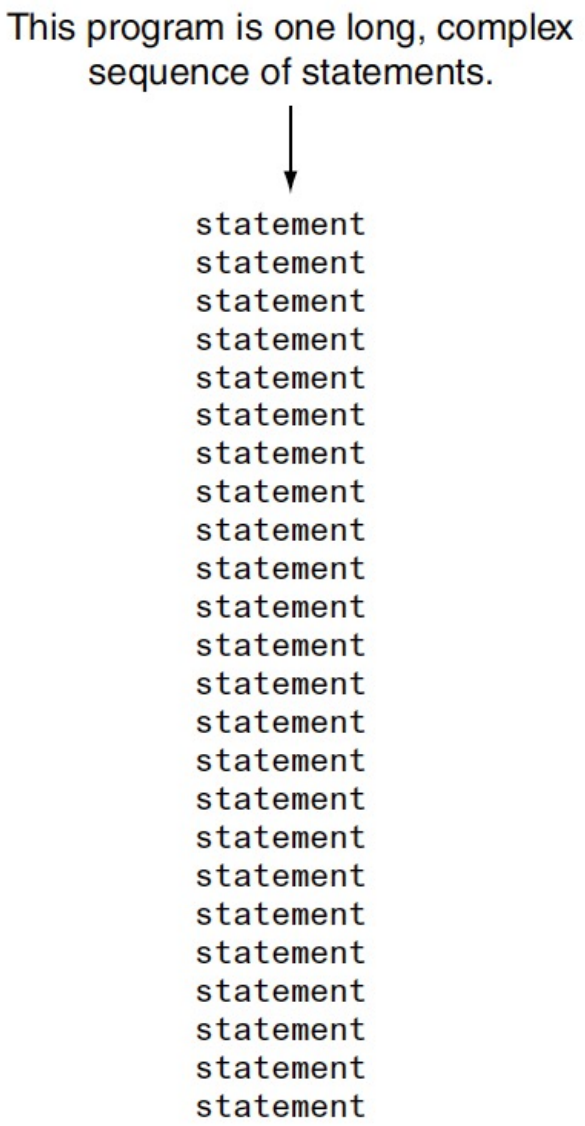
Functions

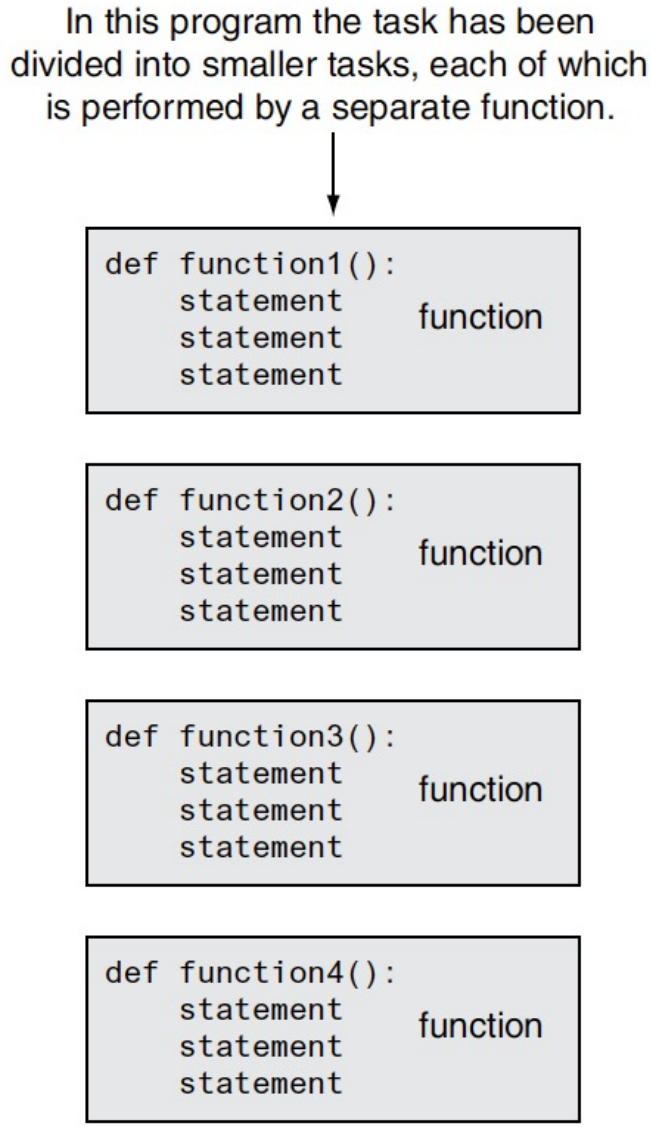
Functions

- Real-life problems are usually complex
- Their solutions may require writing thousands lines of codes (even more)
- Nobody can write that much code at once and from scratch
- Thus, we should divide a large programming task into smaller (and typically easier) subtasks and attack each subtask separately
- This approach is called ***'divide-and-conquer'***
- In programming, functions are very effective tools to facilitate the divide-and-conquer approach

Functions

- Group of statements to perform a specific task (or a subtask)
- Facilitate the divide-and-conquer approach
- Other benefits:
  - Simpler codes
  - Code reuse
  - Easier to debug and test
  - Faster development
  - Easier facilitation of teamwork

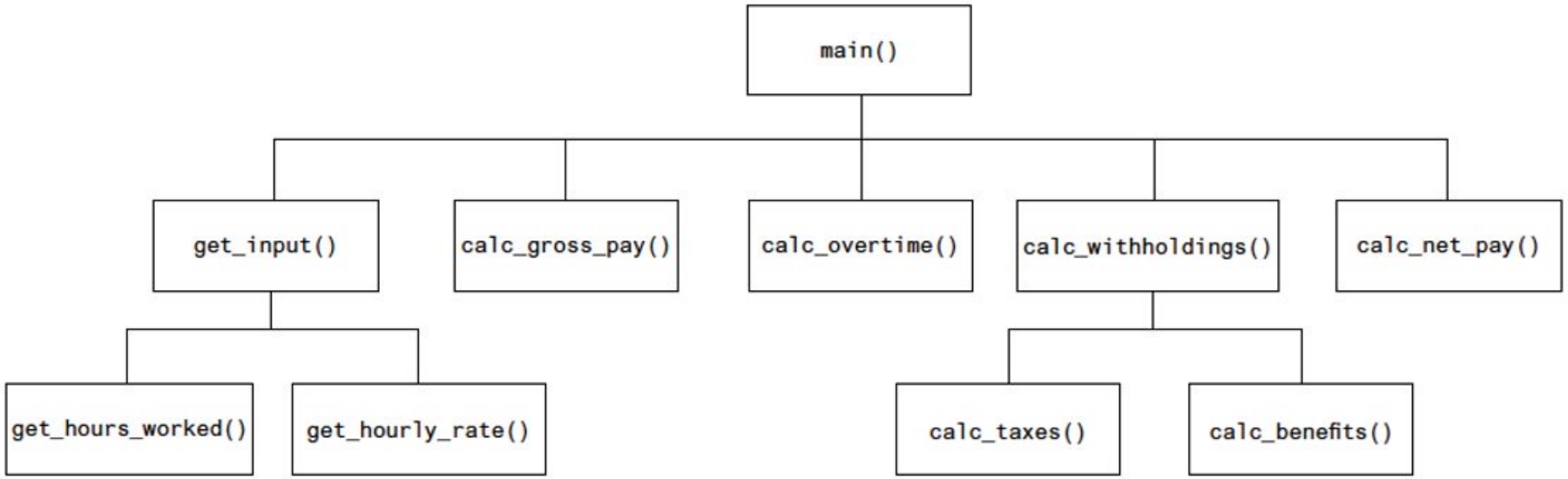




Top-Down Design and Hierarchy Charts

- Overall task is divided into subtasks
- Each subtask is further investigated to see whether they can be broken down into further subtasks
- Write the code accordingly

Figure 5-10 A hierarchy chart



What does it mean? An example

- Write a program that asks the user to enter the following personal information
  - First name
  - Last name
  - Date of birth
  - Place of birth
  - Father name
  - Mother name
  - Blood type
  - Annual income
- Your program should display the following warning message after asking each of these from the user

```
-----------------------------------
This information is confidential
We will not share it with the third parties
We will keep this information in a secure computer
-----------------------------------
```

```
print('Enter your first name')
print('-----------------------------------')
print('This information is confidential')
print('We will not share it with the third parties')
print('We will keep this information in a secure computer')
print('-----------------------------------')
first_name = input()


print('Enter your last name')
print('-----------------------------------')
print('This information is confidential')
print('We will share it with the third parties')
print('We will keep this information in a secure computer')
print('-----------------------------------')
last_name = input()


print('Enter your date of birth')
print('-----------------------------------')
print('This information is confidential')
print('We will not share it with the third parties')
print('We will keep this information in a secure computer')
print('-----------------------------------')
date_of_birth = input()


print('Enter your place of birth')
print('-----------------------------------')
print('This information is confidential')
print('We will not share it with the third parties')
print('We will keep this information in a secure computer')
print('-----------------------------------')
place_of_birth = input()


print('Enter your father name')
print('-----------------------------------')
print('This information is confidential')
print('We will not share it with the third parties')
print('We will keep this enformation in a secure computer')
print('-----------------------------------')
father_name = input()


print('Enter your mother name')
print('-----------------------------------')
print('This information is confidential')
print('We will not share it with the third parties')
print('We will keep this information in a secure computer')
print('-----------------------------------')
mother_name = input()
```

```
print('Enter your blood type')
print('-----------------------------------')
print('This information is confidential')
print('We will not share it with the third parties')
print('We will keep this information in a secure Computer')
print('-----------------------------------')
blood_type = input()


print('Enter your annual income')
print('-----------------------------------')
print('This information is confidential')
print('We will not share it with the third parties')
print('We will keep this information in a seecure computer')
print('-----------------------------------')
annual_income = input()
```

The same statements repeat many times

- Rewrite the same statements every time you need
- Longer program
- Need to test each of them separately

We can do better!

```
def confidentiality_message():
    print('------------------------------------')
    print('This information is confidential')
    print('We will not share it with the third parties')
    print('We will keep this information in a secure computer')
    print('------------------------------------')


def main():
    print('Enter your first name')
    confidentiality_message()
    first_name = input()

    print('Enter your last name')
    confidentiality_message()
    last_name = input()

    print('Enter your date of birth')
    confidentiality_message()
    date_of_birth = input()

    print('Enter your place of birth')
    confidentiality_message()
    place_of_birth = input()

    print('Enter your father name')
    confidentiality_message()
    father_name = input()

    print('Enter your mother name')
    confidentiality_message()
    mother_name = input()

    print('Enter your blood type')
    confidentiality_message()
    blood_type = input()

    print('Enter your annual income')
    confidentiality_message()
    annual_income = input()


main()
```

We can even do much better!

```
def confidentiality_message():
    print('------------------------------------')
    print('This information is confidential')
    print('We will not share it with the third parties')
    print('We will keep this information in a secure computer')
    print('------------------------------------')


def get_input(message_str):
    print(message_str)
    confidentiality_message()
    user_input = input()
    return user_input

def main():
    first_name = get_input('Enter your first name')
    last_name = get_input('Enter your last name')
    date_of_birth = get_input('Enter your date of birth')
    place_of_birth = get_input('Enter your place of birth')
    father_name = get_input('Enter your father name')
    mother_name = get_input('Enter your mother name')
    blood_type = get_input('Enter your blood type')
    annual_income = get_input('Enter your annual income')

main()
```

Remember the built-in range() function, now you will learn how to write your own functions (user-defined functions) and how to call them in your program.

What does it mean? Another example

- Write a program that asks the user to enter m and k and calculates the number of combinations of k elements out of m elements

$$C(m, k) = m! / (k! (m – k)!)$$

- You may assume that k > 0, m > 0, and m >= k

```
factorial = 1
for i in range(2, N + 1):
    factorial *= i
```

```python
m = int(input('m: '))
k = int(input('k: '))

m_factorial = 1
for i in range(2, m + 1):
    m_factorial *= i

k_factorial = 1
for i in range(2, k + 1):
    k_factorial *= i

m_k_factorial = 1
for i in range(2, m - k + 1):
    m_k_factorial *= i

comb = m_factorial / (k_factorial * m_k_factorial)
```

We wrote the same code three times!!!

```
def factorial(n):
    res = 1
    for i in range(2, n + 1):
        res *= i
    return res

def main():
    m = int(input('m: '))
    k = int(input('k: '))
    comb = factorial(m) / (factorial(k) * factorial(m-k))

main()
```

You can re-use the same code by implementing a function
You will learn how to implement a function and call it in your program

Functions

```
def function_name():
    statement
    statement
    …
```

indentation is required

Function **header**

- Starts with def and ends with a colon
- Parentheses after the function name must be used

Function definition **block**

- Specifies what the function does

Function **name**

- Should be a valid identifier name
- Naming rules are the same with those of variables
  - Must contain a letter, a digit or an underscore character
  - Cannot contain spaces
  - First character cannot be a digit
  - Cannot be a keyword (if, def, while, etc.)

o Python is case-sensitive

Example

- Write a function that displays the following message on the screen

```
-----------------------------------
This information is confidential
We will not share it with the third parties
We will keep this information in a secure computer
-----------------------------------
```

```
def confidentiality_message():
    print('-----------------------------------')
    print('This information is confidential')
    print('We will not share it with the third parties')
    print('We will keep this information in a secure computer')
    print('-----------------------------------')
```

How to execute a function?

- **It should be called**

```
def confidentiality_message():
    print('-----------------------------------')
    print('This information is confidential')
    print('We will not share it with the third parties')
    print('We will keep this information in a secure computer')
    print('-----------------------------------')

confidentiality_message()
```

This statement calls the `confidentiality_message` function,
which causes this function to start executing

How to execute a function?

- You may have multiple functions in a program
- How to achieve the *mainline logic* within the program?
  - By using the **main** function

```
def myfunc1():
    print('Executing function 1...')

def myfunc2():
    print('Executing function 2...')

def main():
    print('Starting with the main...')
    myfunc1()
    myfunc2()
    myfunc1()
    print('Ending main now...')

main()
```

```
Starting with the main...
Executing function 1...
Executing function 2...
Executing function 1...
Ending main now...
```

- **Caller** is a function that calls another function
- **Callee** is a function that is called

Exercise

- What is the output of this program?

```
def func1():
    print('1')
    func2()

def func2():
    print('2')
    func3()

def func3():
    print('3')

def main():
    func1()
    print('---')
    func3()
    print('---')
    func1()
    print('---')
    func2()

main()
```

How do functions communicate?

- Functions communicate with each other by passing information
- The caller function passes information to the callee via the parameter list (passing arguments for the parameters)
- The callee function passes information to the caller via the return values

```
def factorial(n):
    res = 1
    for i in range(2, n + 1):
        res *= i
    return res

def main():
    m = int(input('m: '))
    k = int(input('k: '))
    comb = factorial(m) / (factorial(k) * factorial(m-k))

main()
```

A function can use its own variables, which are called **local variables**. This prevents unintended side effects.

res and i are the local variables of the factorial function and m, k, and comb are the local variables of the main function.

Local Variables

- Local variables are created inside a function
- **Variable's scope:** A part of the program in which a variable can be referenced
  - Local variables can only be referenced (accessed) inside a function where they were created
  - You may have variables with the same name in different functions. They will be referenced according to their scopes.

```
def main():
    texas()
    california()

def texas():
    birds = 5000
    print('Texas has', birds, 'birds.')

def california():
    birds = 8000
    print('California has', birds, 'birds.')

main()
```

```
In [6]: runcell(2, '/Users/berensemiz/Desktop/
Comp125 - Practice/python_practice4.py')
Texas has 5000 birds.
California has 8000 birds.
```

Exercise

- What is the output of this program?

```
def foo():
    n = 10
    m = int(input('m: '))
    print(n, m)
    n *= m
    m += 1
    print(n, m)

def bar():
    n = 'COMP'
    course_code = input('Course code: ')
    n += course_code
    print(n)

def main():
    foo()
    foo()
    bar()
    bar()

main()
```

Exercise

- What is the output of this program?

```
def func1():
    print('---func1 starts')
    a = 10
    print(a)
    print('---func1 ends')

def func2():
    print('---func2 starts')
    a = 20
    func1()
    print(a)
    print('---func2 ends')

def main():
    print('---main starts')
    a = 30
    func2()
    print(a)
    func1()
    print(a)
    print('---main ends')

main()
```

Exercise

- What is the output of this program?

```
# Definition of the main function.
def main():
    get_name()
    print('Hello', name) # This causes an error!

def get_name():
    name = input('Enter your name: ')

# Call the main function.
main()
```

```
In [4]: runcell(1, '/Users/berensemiz/Desktop/
Comp125 - Practice/python_practice4.py')

Enter your name: Beren
Traceback (most recent call last):

  File "/Users/berensemiz/Desktop/Comp125 -
Practice/python_practice4.py", line 20, in
<module>
    main()

  File "/Users/berensemiz/Desktop/Comp125 -
Practice/python_practice4.py", line 14, in main
    print('Hello', name) # This causes an error!

NameError: name 'name' is not defined
```

You cannot refer (access) a local variable of one function from the other one

Passing Arguments to Functions

- **Parameter(s):** One or more variables that the callee function expects as input
- **Argument(s):** Values passed by the caller to the callee function and assigned to the parameter(s) of the callee

```
def main():
    value = 5
    show_double(value)

def show_double(number):
    result = number * 2
    print('When we double',number,', we get',result)

main()
```

argument

parameter

```
In [7]: runcell(3, '/Users/berensemiz/Desktop/
Comp125 - Practice/python_practice4.py')
When we double 5 , we get 10
```

Passing Arguments to Functions

```
def main():
    value = 5
    show_double(value)

def show_double(number):
    result = number * 2
    print('When we double',number,', we get',result)

main()
```

argument

parameter

```
In [7]: runcell(3, '/Users/berensemiz/Desktop/
Comp125 - Practice/python_practice4.py')
When we double 5 , we get 10
```

Figure 5-13 The value variable is passed as an argument

```
def main():
    value = 5
    show_double(value)

def show_double(number):
    result = number * 2
    print(result)
```

Passing Multiple Arguments

```
def main():
    print('The sum of 20 and 35 is')
    show_sum(20, 35)


def show_sum(num1, num2):
    result = num1 + num2
    print(result)

main()
```


```
In [8]: runcell(3, '/Users/berensemiz/Desktop/
Comp125 - Practice/python_practice4.py')
The sum of 20 and 35 is
55
```

Passing Multiple Arguments

```
def main():
    print('The sum of 20 and 35 is')
    show_sum(20, 35)


def show_sum(num1, num2):
    result = num1 + num2
    print(result)

main()
```

Arguments are passed by position

```
In [8]: runcell(3, '/Users/berensemiz/Desktop/
Comp125 - Practice/python_practice4.py')
The sum of 20 and 35 is
55
```

Exercise

- Write a function that takes the course code in terms of the department abbreviation and the course number and displays a welcome message

```
def welcome(dept, no):
    print('Welcome to', dept, no)
```

```
def main():
    welcome('COMP', 125)
    welcome('COMP', 448)
    welcome('MATH', 101)
```

How to call this function?

Making Changes to Parameters

```
def main():
    value=99
    print('The value is', value)
    change_me(value)
    print('Back in main the value is', value)

def change_me(arg):
    print('I am changing the value.')
    arg=0
    print('Now the value is', arg)

main()
```

```
In [9]: runcell(6, '/Users/berensemiz/Desktop/
Comp125 - Practice/python_practice4.py')
The value is 99
I am changing the value.
Now the value is 0
Back in main the value is 99
```

Keyword Arguments

- Instead of passing the arguments by position, you can also pass them as

parameter_name = value

```
def main():
    show_interest(rate=0.01, periods=10, principal=10000.0)

def show_interest(principal, rate, periods):
    interest = principal * rate * periods
    print('The simple interest will be $',format(interest, '.2f'))

main()
```

```
In [11]: runcell(8, '/Users/berensemiz/Desktop/
Comp125 - Practice/python_practice4.py')
The simple interest will be $ 1000.00
```

Keyword Arguments

- What if ...

```
def main():
    show_interest(10000.0, rate=0.01, periods=10)

def show_interest(principal, rate, periods):
    interest = principal * rate * periods
    print('The simple interest will be $',format(interest, '.2f'))

main()
```

```
In [12]: runcell(8, '/Users/berensemiz/Desktop/
Comp125 - Practice/python_practice4.py')
The simple interest will be $ 1000.00
```

```
def main():
    show_interest(10000.0, rate=0.01, 10)

def show_interest(principal, rate, periods):
    interest = principal * rate * periods
    print('The simple interest will be $',format(interest, '.2f'))

main()
```

```
In [13]: runcell(8, '/Users/berensemiz/Desktop/
Comp125 - Practice/python_practice4.py')
  File "/Users/berensemiz/Desktop/Comp125 -
Practice/python_practice4.py", line 104
    show_interest(10000.0, rate=0.01, 10)
                                      ^
SyntaxError: positional argument follows keyword
argument
```

Positional arguments must appear first, followed by the keyword arguments.

Exercise

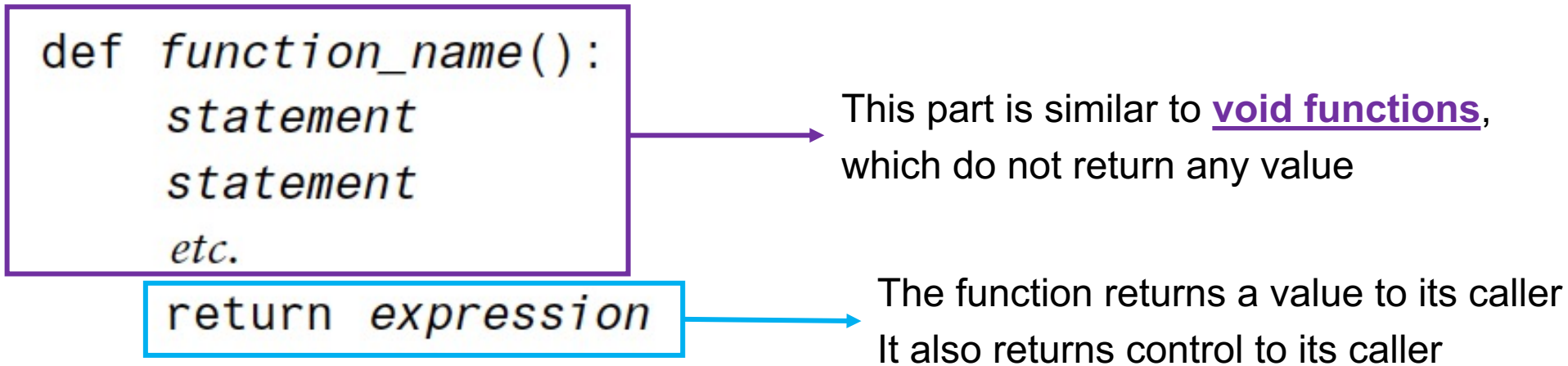
- What is the output of this program?

```
def func(a, b, c, d):
    print(a, b, c, d, sep = ' - ')

def main():
    func(1, 2, 3, 4)
    func(1, c = 2, d = 3, b = 4)
    func(1, 2, d = 3, c = 4)
    func(1, 2, a = 3, d = 4)

main()
```

```
In [1]: runfile('/Users/cigdem/Desktop/comp125/py Files
06_output4.py', wdir='/Users/cigdem/Desktop/comp125/py
1 - 2 - 3 - 4
1 - 4 - 2 - 3
1 - 2 - 4 - 3
Traceback (most recent call last):

  File "/Users/cigdem/Desktop/comp125/py Files/06_outpu
line 10, in <module>
    main()

  File "/Users/cigdem/Desktop/comp125/py Files/06_outpu
line 8, in main
    func(1, 2, a = 3, d = 4)

TypeError: func() got multiple values for argument 'a'
```

Value Returning Functions

- A function may pass information to its caller function by returning values



Example

- Write a function that takes two numbers as its inputs and returns their sum

```
def calculate_sum(first, second):
    result = first + second
    return result
```

```
def main():
    a = 5
    sum1 = calculate_sum(3, a)
    sum2 = calculate_sum(a * 7, a)
    print(sum1)
    print(sum2)

main()
```

One common mistake!

- Many students use `print()` instead of `return`
- `print()` does not pass any information to the caller
- That is, the caller cannot access/use the value

```
def show_sum(first, second):
    result = first + second
    print(result)

def main():
    a = 5
    show_sum(3, a)
```

The show_sum function calculates the sum of first and second, which is 8, and displays it on the screen. However, the main function cannot access this sum and use it.

Thus, if the caller function needs to use this value, you have to use return.

WARNING: You will lose points from this mistake in the exams and the assignments!!!

return

- Immediately returns control to the caller function
- Causes to terminate the callee function

Example: What is the output?

```
def func(a):
    print('Function starts')
    print(a)
    a = 5
    print('Before return')
    return a
    print('After return')
    a = 20

def main():
    a = 10
    b = func(a)
    print(a)
    print(b)

main()
```

Exercise

- Now, let's write the program that asks the user to enter m and k and calculates the number of combinations of k elements out of m elements

$$C(m, k) = m! / (k! (m – k)!)$$

- Add the validity check on the inputs (k > 0, m > 0, and m >= k)

```python
def factorial(n):
    res = 1
    for i in range(2, n + 1):
        res *= i
    return res

def combination(m, k):
    comb = factorial(m) / (factorial(k) * factorial(m-k))
    return comb

def is_valid(m, k):
    return (k > 0) and (m > 0) and (m >= k)

def main():
    m = int(input('m: '))
    k = int(input('k: '))
    if is_valid(m, k):
        print('C(m, k) = ', int(combination(m, k)))
    else:
        print('Invalid inputs')

main()
```

Returning Multiple Values

- In Python, a function can return multiple values
- Specified after the return statements, separated by commas

return sum, average, median

- In the caller function, you need separate variables on the LHS of the assignment operator to receive each returned value

s, a, m = func1(parameters)

```
def example(a):
    return a + 1, a + 2, a + 3

def main():
    r1, r2, r3 = example(3)
```

Exercise

- Write a function that takes the name, semester, and the GPA of the user and returns them to the caller function

```
def get_input():
    name = input('Your name: ')

    semester = input('Your semester: ')
    semester = int(semester)

    gpa = input('Your semester: ')
    gpa = float(semester)

    return name, semester, gpa

def main():
    n, s, g = get_input()

main()
```

Exercise

- Write a function that takes the width and length of a rectangle as inputs and returns the area and perimeter of this rectangle as outputs. This function should return 0 for the area and perimeter if the width or the length is an invalid number.
- Then write the main function that takes the width and length from the user, calls the first function, and displays the area and perimeter on the screen.

```
def rectangle_features(W, L):
    if W <= 0 or L <= 0:
        area = 0
        perimeter = 0
    else:
        area = W * L
        perimeter = 2 * (W + L)
    return area, perimeter

def main():
    width = float(input('Width: '))
    length = float(input('Length: '))
    area, perimeter = rectangle_features(width, length)
    print(area, perimeter)

main()
```

Exercise

- Remember the following prime number example, which was given in the previous slide set
- Now implement it using functions

```
N = int(input('Number: '))
prime = True

divisor = 2
while divisor < N:
    if N % divisor == 0:
        prime = False
    divisor = divisor + 1

if prime:
    print(N, 'is prime')
else:
    print(N, 'is not prime')
```

```
def is_prime(N):
    divisor = 2
    while divisor < N:
        if N % divisor == 0:
            return False
        divisor = divisor + 1

    return True

def main():
    N = int(input('Number: '))
    if N <= 0:
        print(N, 'is invalid input')
    elif is_prime(N):
        print(N, 'is prime')
    else:
        print(N, 'is not prime')

main()
```

Exercise

- What are the outputs?

```
def main():
    x = 2.2
    print(x)
    change_me(x)
    print(x)

def change_me(a):
    a = 0
    print(a)

main()
```

```
2.2
0
2.2
```

```
def main():
    x = 2.2
    print(x)
    x=change_me(x)
    print(x)

def change_me(a):
    a = 0
    print(a)
    return a

main()
```

```
2.2
0
0
```

Exercise

- What are the outputs?

```
def main():
    x = 2.2
    print(x)
    x=change_me(x)
    print(x)

def change_me(a):
    a = 0
    print(a)

main()
```

```
2.2
0
None
```

```
def main():
    x = 2.2
    print(x)
    change_me(x)
    print(a)

def change_me(a):
    a = 0
    print(a)

main()
```

```
NameError: name 'a' is not defined
```

Optional (default) Arguments

It is possible to assign default values to function parameters if the caller omits the corresponding argument.

```
def power(n, power=2):
    return n ** power

print(power(2))
print(power(2,3))
```

Output

```
4
8
```

Optional (default) Arguments

Optional (default) arguments, cannot be followed by Required (non-default) arguments.

```
def power(power=2, n):
    return n ** power
```

Output

```
    def power(power=2, n):
              ^
SyntaxError: non-default argument follows
default argument
```

Excercise

Write a function called *draw_triangle (height, pattern)* which displays a right isosceles triangle based on the parameters passed. If pattern is not provided the triangle is drawn with * character.

- Examples

draw_triangle(6) displays:

```
*
**
***
****
*****
******
```

draw_triangle(3, '.') displays:

```
.
. .
. . .
```

Excercise

```
def draw_triangle(height, pattern = '*'):
    for i in range(height):
        print(pattern * (i+1))

def main():
    draw_triangle(5)
    draw_triangle(3, '.')


main()
```

Modules

Standard Library

- Library of the pre-implemented functions that come with Python
- Perform tasks that programmers commonly need
  - e.g., **`print`**, **`input`**, **`range`**, …
  - We commonly use them as black boxes
- https://docs.python.org/3/library/

import + module

- **Modules**:
  - Files that store existing code and functions
  - To call a function stored in a module, you need to import it
    - Written at the top of the program, `import module_name`

```
import math
import random
...
```

The math module

- Contains many different functions that can be used in mathematical calculations
- You should **import** the **`math`** module prior to calling these functions

```
import math

def main():
    number = float(input('Enter a number: '))
    square_root = math.sqrt(number)
    print('The square root of', number, 'is', square_root)

main()
```

```
Enter a number: 4
The square root of 4.0 is 2.0
```

The math module

Table 5-2 Many of the functions in the `math` module

`math` Module Function	Description
`acos(x)`	Returns the arc cosine of `x`, in radians.
`asin(x)`	Returns the arc sine of `x`, in radians.
`atan(x)`	Returns the arc tangent of `x`, in radians.
`ceil(x)`	Returns the smallest integer that is greater than or equal to `x`.
`cos(x)`	Returns the cosine of `x` in radians.
`degrees(x)`	Assuming `x` is an angle in radians, the function returns the angle converted to degrees.
`exp(x)`	Returns e^x
`floor(x)`	Returns the largest integer that is less than or equal to `x`.
`hypot(x, y)`	Returns the length of a hypotenuse that extends from (0, 0) to (`x`, `y`).
`log(x)`	Returns the natural logarithm of `x`.
`log10(x)`	Returns the base-10 logarithm of `x`.
`radians(x)`	Assuming `x` is an angle in degrees, the function returns the angle converted to radians.
`sin(x)`	Returns the sine of `x` in radians.
`sqrt(x)`	Returns the square root of `x`.
`tan(x)`	Returns the tangent of `x` in radians.

All return a `float` value except **`ceil`** and **`floor`** functions (these return `int`!)

The math module

- The math module defines two **variables:**
  - math.pi (~3.141592653589793)
  - math.e (~2.718281828459045)

```
import math

def main():
    my_radius=float(input('Enter a radius value: '))
    print('The area is', area(my_radius))

def area(radius):
    area = math.pi * radius**2
    return area

main()
```

```
Enter a radius value: 3
The area is 28.274333882308138
```

Storing Functions as Modules

- You can create your own module

circle.py

```
1   import math
2
3   def area(radius):
4       area = math.pi * radius**2
5       return area
6
7   def perimeter(radius):
8       peri = 2 * math.pi * radius
9       return peri
10
```

calculations.py

```
1   import circle
2
3   def main():
4       my_radius = float(input('Enter a radius: '))
5       my_area = circle.area(my_radius)
6       my_perimeter = circle.perimeter(my_radius)
7       print('The area is', my_area)
8       print('The perimeter is', my_perimeter)
9
10  main()
```

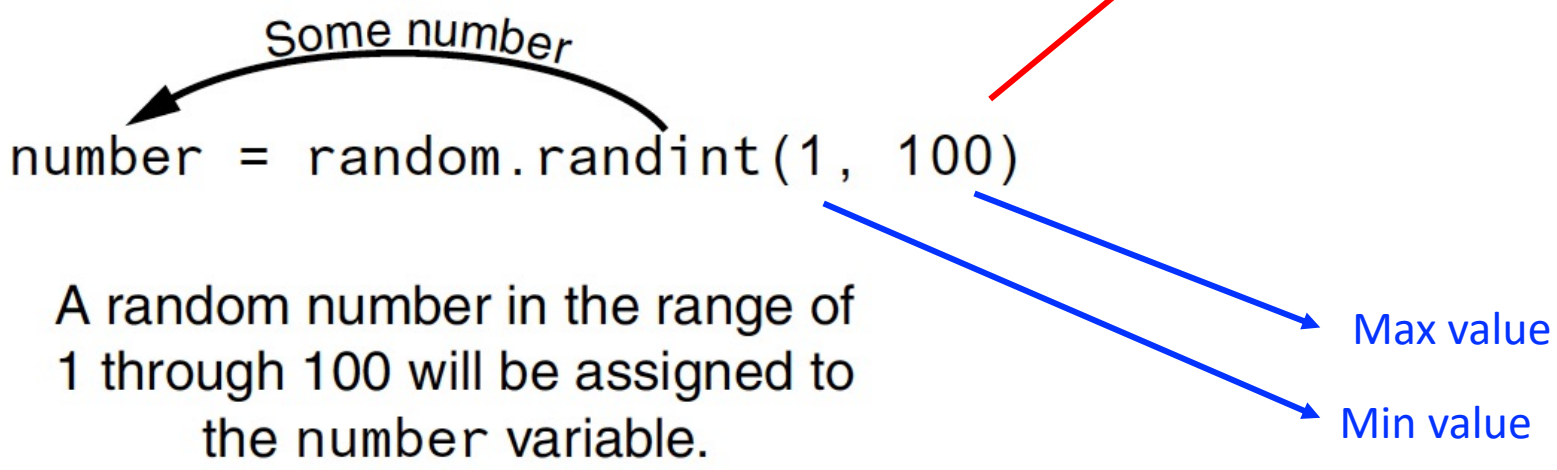
```
Enter a radius: 5
The area is 78.53981633974483
The perimeter is 31.41592653589793
```

The random module

- Provides several functions for working with random numbers

```
import random
```

BOTH ARE INCLUDED!!!



Example

- Backgammon time!
- Write a program to simulate the rolling of dice. The program should randomly generate two numbers in the range of 1 through 6 and display them. Program should keep rolling the dice as long as the user wants to continue.
- Pseudocode:

While the user wants to roll the dice

Display a random number in the range of 1 through 6

Display another random number in the range of 1 through 6

Ask the user if he or she wants to roll the dice again

Example

- You are asked to write a program to simulate ten coin tosses, one after the other. Each time the program simulates a coin toss, it should randomly display either **“Heads”** or **“Tails”**. You decide that you can simulate the tossing of a coin by randomly generating a number.
- Pseudocode:

```
Repeat 10 times
    If a random number in the range of 1 through 2 equals 1 then
        Display ‘Heads’
    Else
        Display ‘Tails’
```

The random module

- **`randrange:`** returns a randomly selected value within the specified range
- **`random:`** returns a random **floating-point** number between 0.0 and 1.0
- **`uniform:`** returns a random **floating-point** number, you can specify the range

```
import random

number1 = random.randrange(10)
print('Number1 is:',number1)

number2 = random.randrange(0,101,10)
print('Number2 is:',number2)

number3 = random.random()
print('Number3 is:',number3)

number4 = random.uniform(1.0,10.0)
print('Number4 is:',number4)
```

0 through 9 (10 not included)

0 through 100 with a step-size of 10 (101 not included)

You don't need to specify. Between 0.0 - 1.0 (1.0 not included)

Between 1.0 - 10.0 (10.0 included)

```
Number1 is: 4
Number2 is: 90
Number3 is: 0.18890849305216106
Number4 is: 8.915886700815363
```

The random number seeds

- The random numbers we are receiving are actually *pseudorandom* numbers calculated by a formula
- This formula is initialized with a value known as **seed value**
- Can use `random.seed(value)` to specify the desired seed value

Let’s practice

TRUE OR FALSE?

- You do not need to have an import statement in a program to use the functions in the random module
- **FALSE**
- A function in Python can return more than one value
- **TRUE**
- A statement in one function can access a local variable in another function
- **FALSE**