Session 5

What is a Module?

Consider a module to be the same as a code library.

A file containing a set of functions you want to include in your application.

Create a Module

To create a module just save the code you want in a file with the file extension `.py`:

PYTHON

```python
# Save this code in a file named `mymodule.py`

def greeting(name):
  print("Hello, " + name)
```

Use a Module

Now we can use the module we just created, by using the `import` statement:

PYTHON

```python
# Import the module named mymodule, and call the greeting function:
import mymodule

mymodule.greeting("Jonathan")
```

> **Note:** When using a function from a module, use the syntax: *module_name.function_name*.

Variables in Module

The module can contain functions, as already described, but also variables of all types (arrays, dictionaries, objects etc):

PYTHON

```python
# Save this code in the file `mymodule.py`

person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

PYTHON

```python
# Import the module named mymodule, and access the person1
dictionary:

import mymodule

a = mymodule.person1["age"]
print(a)
```

Naming a Module

You can name the module file whatever you like, but it must have the file extension `.py`

Re-naming a Module

You can create an alias when you import a module, by using the `as` keyword:

PYTHON

```python
# Create an alias for `mymodule` called `mx`:

import mymodule as mx

a = mx.person1["age"]
print(a)
```

Import From Module

You can choose to import only parts from a module, by using the `from` keyword.

PYTHON

```python
# The module named `mymodule` has one function and one dictionary:

def greeting(name):
  print("Hello, " + name)

person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

PYTHON

```python
# Import only the person1 dictionary from the module:

from mymodule import person1

print (person1["age"])
```

> **Note:** When importing using the `from` keyword, do not use the module name when referring to elements in the module.
> Example: `person1["age"]`, **not** ~~`mymodule.person1["age"]`~~

Python Dates

A date in Python is not a data type of its own, but we can import a module named `datetime` to work with dates as date objects.

PYTHON

```python
# Import the datetime module and display the current date:

import datetime

x = datetime.datetime.now()
print(x)
```

Date Output

When we execute the code from the example above the result will be:

```
2025-02-28 21:18:57.481264
```

The date contains year, month, day, hour, minute, second, and microsecond.

The `datetime` module has many methods to return information about the date object.

Here are a few examples, you will learn more about them later in this chapter:

PYTHON

```python
# Return the year and name of weekday:

import datetime

x = datetime.datetime.now()

print(x.year)
print(x.strftime("%A"))
```

Creating Date Objects

To create a date, we can use the `datetime()` class (constructor) of the `datetime` module.

The `datetime()` class requires three parameters to create a date: year, month, day.

Create a date object:

PYTHON

```python
import datetime

x = datetime.datetime(2020, 5, 17)

print(x)
```

The `datetime()` class also takes parameters for time and timezone (hour, minute, second, microsecond, tzone), but they are optional, and has a default value of `0`, (`None` for timezone).

The `strftime()` Method

The `datetime` object has a method for formatting date objects into readable strings.

The method is called `strftime()`, and takes one parameter, `format`, to specify the format of the returned string:

```python
# Display the name of the month:

import datetime

x = datetime.datetime(2018, 6, 1)

print(x.strftime("%B"))
```

A reference of all the legal format codes:

Directive	Description	Example
%a	Weekday, short version	Wed
%A	Weekday, full version	Wednesday
%w	Weekday as a number 0-6, 0 is Sunday	3
%d	Day of month 01-31	31
%b	Month name, short version	Dec
%B	Month name, full version	December
%m	Month as a number 01-12	12
%y	Year, short version, without century	18
%Y	Year, full version	2018
%H	Hour 00-23	17
%I	Hour 00-12	05
%p	AM/PM	PM
%M	Minute 00-59	41
%S	Second 00-59	08
%f	Microsecond 000000-999999	548513
%z	UTC offset	+0100
%Z	Timezone	CST
%j	Day number of year 001-366	365
%U	Week number of year, Sunday as the first day of week, 00-53	52
%W	Week number of year, Monday as the first day of week, 00-53	52
%c	Local version of date and time	Mon Dec 31 17:41:00 2018
%C	Century	20

Directive	Description	Example
%x	Local version of date	12/31/18
%X	Local version of time	17:41:00
%%	A % character	%
%G	ISO 8601 year	2018
%u	ISO 8601 weekday (1-7)	1
%V	ISO 8601 weeknumber (01-53)	01

Python Math

Python has a set of built-in math functions, including an extensive math module, that allows you to perform mathematical tasks on numbers.

Built-in Math Functions

The `min()` and `max()` functions can be used to find the lowest or highest value in an iterable:

PYTHON

```
x = min(5, 10, 25)
y = max(5, 10, 25)

print(x)
print(y)
```

The `abs()` function returns the absolute (positive) value of the specified number:

PYTHON

```
x = abs(-7.25)

print(x)
```

The `pow(_x_, _y_)` function returns the value of x to the power of y (x ^ y).

Return the value of 4 to the power of 3 (same as 4 *4* 4):

```python
x = pow(4, 3)

print(x)
```

The Math Module

Python has also a built-in module called `math`, which extends the list of mathematical functions.

To use it, you must import the `math` module:

```python
import math
```

When you have imported the `math` module, you can start using methods and constants of the module.

The `math.sqrt()` method for example, returns the square root of a number:

```python
import math

x = math.sqrt(64)

print(x)
```

The `math.ceil()` method rounds a number upwards to its nearest integer, and the `math.floor()` method rounds a number downwards to its nearest integer, and returns the result:

Example

PYTHON

```python
import math

x = math.ceil(1.4)
y = math.floor(1.4)

print(x) # returns 2
print(y) # returns 1
```