Python Programming 1

(CP1S02005E)

Białystok University of Technology
Faculty of Electrical Engineering

Industry Digitization, semester II
Academic year 2024/2025

Lecture no. 11 (28.05.2025)

Jarosław Forenc, PhD

Topics

- Python Standard Library
  - built-in functions
  - built-in constants
  - built-in types
  - built-in exceptions
  - modules (e.g., string, datetime, zoneinfo, calendar)

Python - Standard Library

- a collection of modules and packages distributed with the core installation of the Python language
- documentation: https://docs.python.org/3/library/index.html
- the library is extensive and includes many well-tested and efficient functions and modules
- the library includes modules:
  - built-in, written in C - providing access to system-level functionality, such as file operations
  - written in Python - offering standard solutions for many problems encountered during everyday programming
- on Windows, the Python installer includes the full standard library and many additional components
- on Linux/Unix, Python is distributed as a collection of packages, so it may be necessary to install some components manually

Python - Standard Library

- the standard library includes:
  - built-in functions and constants
  - built-in types and exceptions
  - modules
- example purposes of modules:
  - text and binary data processing services, data types
  - numerical and mathematical modules
  - file and directory access, data compression and archiving
  - cryptographic services, general OS services
  - networking and interprocess communication, internet data handling
  - internet protocols and support, multimedia services
  - internationalization, developer tools, module importing

Python - list of built-in functions

A

abs()
aiter()
all()
anext()
any()
ascii()

B

bin()
bool()
breakpoint()
bytearray()
bytes()

C

callable()
chr()
classmethod()
compile()
complex()

D

delattr()
dict()
dir()
divmod()

E

enumerate()
eval()
exec()

F

filter()
float()
format()
frozenset()

G

getattr()
globals()

H

hasattr()
hash()
help()
hex()

I

id()
input()
int()
isinstance()
issubclass()
iter()

Python - list of built-in functions

L

len()
list()
locals()

M

map()
max()
memoryview()
min()

N

next()

O

object()
oct()
open()
ord()

P

pow()
print()
property()

R

range()
repr()
reversed()
round()

S

set()
setattr()
slice()
sorted()
staticmethod()
str()
sum()
super()

T

tuple()
type()

V

vars()

Z

zip()

_

__import__()

Python - list of built-in constants

- True - represents the Boolean value true (bool type)
- False - represents the Boolean value false (bool type)
- None - represents the absence of a value or null; used when, for example, default arguments are not provided to a function (NoneType)
- NotImplemented - a special value used to indicate that a method is not implemented for certain data types
- Ellipsis - represented by ... (three dots); mainly used in special cases like slicing in multidimensional arrays
- debug - a constant that is True when Python runs in debug mode
- copyright, credits, license - informational constants displaying Python's copyright, credits, and license

Python - list of built-in constants

```
print(copyright)
print(credits)
print(license)
```

```
Copyright (c) 2001-2023 Python Software Foundation.
All Rights Reserved.

Copyright (c) 2000 BeOpen.com.
All Rights Reserved.

Copyright (c) 1995-2001 Corporation for National Research
Initiatives.
All Rights Reserved.

Copyright (c) 1991-1995 Stichting Mathematisch Centrum,
Amsterdam.
All Rights Reserved.
    Thanks to CWI, CNRI, BeOpen.com, Zope Corporation and a cast
of thousands
    for supporting Python development.  See www.python.org for
more information.
Type license() to see the full license text
```

Python - list of built-in types

- numeric types:
  - int - integers, arbitrarily large (no size limit), e.g., a = 23
  - float - floating-point numbers, similar to double in C, e.g., b = 2.5
  - complex - complex numbers with real and imaginary parts, e.g., z = 2 + 5j
- sequence types:
  - list - a dynamic array of elements of any type, mutable - its content can be changed after creation, e.g., lst = [1, 2, 3]
  - tuple - an immutable sequence of elements - cannot be changed after creation, e.g., t = (1, 2, 3, 'x', 'y', 'z')
  - range - lazily generates a sequence of integers (numbers are generated on demand, not stored in memory), e.g., r = range(100)
- text type:
  - str - a string of characters (text); uses Unicode encoding, e.g., txt = "Hello"

Python - list of built-in types

- binary types:
  - bytes - an immutable sequence of bytes (values 0–255), e.g., b = b'hello'
  - bytearray - a mutable sequence of bytes, e.g., ba = bytearray(b'hello')
  - memoryview - enables efficient operations on large binary objects without copying them, e.g., mv = memoryview(b'hello')
- set types:
  - set - a mutable set of unique elements, e.g., s = {1, 2}
  - frozenset - an immutable set of unique elements, e.g., fs = frozenset([1, 2, 3, 'x', 'y', 'z'])
- mapping type:
  - dict - a dictionary, a collection of key-value pairs where keys are unique, and each key maps to one value, e.g., d = {'k1': 'v1', 'k2': 'v2'}

Python - list of built-in types

- boolean type:
  - bool - represents the boolean values True and False; it is a subtype of int (inherits all of its properties), e.g., b = True
- special type:
  - NoneType - represented by the single object None; signifies the absence of a value or nonexistence, e.g., n = None
- built-in iterable types:
  - enumerate - a function that returns an iterator producing tuples containing the index and the corresponding element from a sequence, e.g., e = enumerate(['x', 'y', 'z'])
  - reversed - a function that returns an iterator that yields the elements of a sequence in reverse order, e.g., r = reversed([6, 7, 8])
  - zip - a function that returns an iterator that aggregates elements from multiple iterables, e.g., z = zip([6, 7, 8], ['x', 'y', 'z'])

Python - list of built-in types

- function types:
  - function - a function defined using the def keyword or a lambda, e.g., def func(x): return x ** 2
  - lambda - an anonymous function with any number of arguments but only one expression, e.g., f = lambda x: x ** 2
- class and object types:
  - class - a class defined using the class keyword, e.g., class MyClass: pass
  - object - the base class for all classes, e.g., o = object()
- exception types:
  - BaseException - the base class for all exceptions
  - Exception - the base class for most built-in exceptions related to common programming errors

Python - list of built-in exceptions

- exceptions are special objects used by Python to handle errors that may occur during program execution

```
try:
    # code that might raise an exception
except ExceptionType:
    # exception handling
finally:
    # code that will always execute
```

```
try:
    # code that might raise an exception
except ExceptionType:
    # exception handling
else:
    # code that runs only if no exception occurred
```

Python - list of built-in exceptions

- Exception - the base class for all built-in exceptions in Python; custom and standard exceptions derive from this class
- BaseException - the ultimate base class for all exceptions, including those not related to typical errors (should not be used directly)
- KeyboardInterrupt - occurs when the user interrupts program execution using a keyboard shortcut (Ctrl+C)
- SystemExit - occurs when the program exits using sys.exit(); catching this prevents the program from closing
- StopIteration - raised by iterators to signal the end of iteration (automatically handled by for loops)
- AttributeError - raised when trying to access an attribute that does not exist on an object
- EOFError - raised when input functions (like input() or read()) reach the end of the file (EOF)

Python - list of built-in exceptions

- ArithmeticError - base class for all arithmetic-related exceptions:
  - ZeroDivisionError - raised when dividing a number by zero
  - OverflowError - raised when the result of an arithmetic operation is too large to be represented
  - FloatingPointError - raised for floating-point operation errors
- ImportError - raised when importing a module fails
  - ModuleNotFoundError - raised when the specified module cannot be found
- IndexError - raised when a sequence index is out of range
- KeyError - raised when a dictionary key is not found
- MemoryError - raised when an operation runs out of memory

Python - list of built-in exceptions

- NameError - raised when a variable or symbol is not defined
- OSError - base class for operating system-related exceptions:
  - FileNotFoundError - raised when a file or directory cannot be found
  - PermissionError - raised when permission is denied to perform an operation
  - IsADirectoryError - raised when a file operation is attempted on a directory
  - NotADirectoryError - raised when a directory operation is attempted on a non-directory object
- TypeError - raised when an operation or function is applied to an object of an inappropriate type
- ValueError - raised when a function receives an argument of the right type but an inappropriate value

Python - string module

- the string module includes various functions and constants useful for string manipulation
- list of constants:
  - string.ascii_letters - contains all alphabetic letters (upper- and lowercase)

```
import string
print(string.ascii_letters)
```

```
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
```

  - string.ascii_lowercase - contains all lowercase alphabetic letters
  - string.ascii_uppercase - contains all uppercase alphabetic letters
  - string.digits - contains all decimal digits (0-9)
  - string.hexdigits - contains all hexadecimal digits (0-9, a-f, A-F)
  - string.octdigits - contains all octal digits (0–7)

Python - string module

- list of constants:
  - string.punctuation - contains all punctuation characters
  - string.printable - contains all characters considered printable, i.e., letters, digits, punctuation, and whitespace
  - string.whitespace - contains all whitespace characters (spaces, tabs, newlines)

```
import string
txt = """In February 2024, 11,567 new PV installations
were created. Their total capacity amounted to 232.50 MW.
The largest group consists of micro-installations: 11,478
units with a total capacity of 100.33 MW."""

digits = [ch for ch in txt if ch in string.digits]
print(f"Number of digits in the text: {len(digits)}")
```

```
Number of digits in the text: 24
```

Python - datetime module

- the datetime module is used for manipulating dates and times; it includes many functions and classes that simplify operations such as creation, comparison, formatting, and arithmetic
- classes in the module:
  - datetime - represents both date and time as a single object
  - date - represents only the date (no time)
  - time - represents only the time (no date)
  - timedelta - represents the difference between two dates or times
- these classes allow arithmetic operations like addition and subtraction with dates and times
- objects of these classes can be compared using comparison operators such as <, <=, >, >=, ==, and !=
- they also provide methods to access individual components like year, month, day, hour, minute, second

Python - datetime module

```
import datetime

specific_date = datetime.datetime(2024, 5, 1, 12, 0, 0)
current_date = datetime.datetime.now()

print("Specific date:")
print("Year:  ", specific_date.year)
print("Month: ", specific_date.month)
print("Day:   ", specific_date.day)

print("\nCurrent date:")
print("Year:  ", current_date.year)
print("Month: ", current_date.month)
print("Day:   ", current_date.day)
print("Hour:  ", current_date.hour)
print("Minute:", current_date.minute)
print("Second:", current_date.second)
```

```
Specific date:
Year:   2024
Month:  5
Day:    1

Current date:
Year:   2025
Month:  5
Day:    27
Hour:   21
Minute: 33
Second: 21
```

Python - zoneinfo module

- the zoneinfo class enables working with time zones (available from Python 3.9)
- it allows the creation of objects representing specific time zones, such as "Europe/Warsaw", "America/New_York", "Asia/Tokyo", etc.
- zoneinfo objects contain information about time offsets, daylight saving time changes, and other settings specific to a time zone
- this class allows for operations like converting between time zones or checking if a datetime belongs to a specific zone
- zoneinfo objects automatically account for daylight saving time changes

Python - zoneinfo module

```
from datetime import datetime
from zoneinfo import ZoneInfo

warsaw_time = datetime(2025, 5, 13, 10, 15,
                       tzinfo=ZoneInfo("Europe/Warsaw"))
print("Time in Warsaw:  ", warsaw_time)

ny_time = warsaw_time.astimezone(ZoneInfo("America/New_York"))
print("Time in New York:", ny_time)

tokyo_time = warsaw_time.astimezone(ZoneInfo("Asia/Tokyo"))
print("Time in Tokyo:   ", tokyo_time)
```

```
Time in Warsaw:   2025-05-13 10:15:00+02:00
Time in New York: 2025-05-13 04:15:00-04:00
Time in Tokyo:    2025-05-13 17:15:00+09:00
```

- for the program to work correctly, the tzdata package must be installed: pip install tzdata

Python - calendar module

- the calendar module allows generating calendars and performing date-related operations
- it enables creating calendars for specific years and months, accessing weekday information, and performing arithmetic on dates
- functions like calendar.month() and calendar.calendar() generate calendars for a specified month or year
- the TextCalendar class can generate calendars in text format and allows customization, such as setting the first day of the week and formatting
- helper functions include calendar.weekday() (returns the weekday for a given date) and calendar.monthrange() (returns the first weekday and number of days in a month)
- the module also defines constants such as calendar.MONDAY, calendar.TUESDAY, etc., which represent days of the week

Python - calendar module

```
import calendar

print("Calendar for the month of May 2025:")
print(calendar.month(2025, 5))
```

```
Calendar for the month of May 2025:
      May 2025
Mo Tu We Th Fr Sa Su
            1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

Python - calendar module

```
import calendar

print("Weekday number for May 16, 2025:")
print(calendar.weekday(2025, 5, 16))

print("Starting weekday and number of days in January
       2025:")
print(calendar.monthrange(2025, 1))

print("Constants representing days of the week:")
print("Monday:", calendar.MONDAY)
print("Tuesday:", calendar.TUESDAY)
```

```
Weekday number for May 16, 2025:
4
Starting weekday and number of days in January 2025:
(calendar.WEDNESDAY, 31)
Constants representing days of the week:
Monday: 0
Tuesday: 1
```

Python - calendar module

```
import calendar

print(calendar.calendar(2025))
```

```
                                  2025

      January                   February                  March
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
       1  2  3  4  5                      1  2                      1  2
 6  7  8  9 10 11 12       3  4  5  6  7  8  9       3  4  5  6  7  8  9
13 14 15 16 17 18 19      10 11 12 13 14 15 16      10 11 12 13 14 15 16
20 21 22 23 24 25 26      17 18 19 20 21 22 23      17 18 19 20 21 22 23
27 28 29 30 31            24 25 26 27 28            24 25 26 27 28 29 30
                                                    31

      April                     May                       June
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
    1  2  3  4  5  6                1  2  3  4                         1
 7  8  9 10 11 12 13       5  6  7  8  9 10 11       2  3  4  5  6  7  8
14 15 16 17 18 19 20      12 13 14 15 16 17 18       9 10 11 12 13 14 15
21 22 23 24 25 26 27      19 20 21 22 23 24 25      16 17 18 19 20 21 22
28 29 30                  26 27 28 29 30 31         23 24 25 26 27 28 29
                                                    30
```

End of lecture no. 11

Thank you for your attention!