Answers and materials

Very easy:

1. How to correctly print `"Hello world!"`?

Correct answer is

```
print("Hello world!")
```

In Python the easiest way to print out a text is to use built-in function `print`. First argument of this function is an object, which we want to print out. It can be a string of characters: object `str` (like in our case) or other type of an object.
In the second answer it will be converted to string of characters.
Using print function we can also print many objects. For example:

```
print('Ala', 'Jan', 'Adam')
```

will print out `Ala Jan Adam`
Print function may also have additional arguments which can specify:

- separator used to separate printed objects
- character to be printed as a last one
- a place where the text should be printed
- a flag affecting text buffering

Because function `print` is built-in you don't need to import it.
In most cases the basic way of usage:

```
print("Tekst do wypisania")
```

will be enough but it is useful to know all the features. In Python 2 `print` was a keyword (statement) such as `return`.
This made possible to print out:

```
print "Tekst"
```

It has been changed in Python 3 and now `print` is only a built-in function.

More detailed information about `print` function in Python 3 is fully described here:
https://docs.python.org/3/library/functions.html#print

Information about differences between Python 3 and 2:
https://docs.python.org/3/whatsnew/3.0.html

Information about `print` function w Python 2:
https://docs.python.org/2.7/reference/simple_stmts.html#print
https://docs.python.org/2.7/library/functions.html#print

2. Which of the following commands will assign a string of characters `"Python"` to a variable `result`?

Correct answer is

```
result = "Py" + "thon"
```

In Python one of the ways to concatenate strings (connect the strings) is to use operator `+`.
It is possible because built-in type `str` implements appropriate methods which make possible to concatenate or index using operators basing also on sequential types.
This type of command is very clear and well understandable. To connect a new string of characters with an existing value (saved in a variable) we can use operator `+=`

```
welcome = "Hello"

welcome += " World!"
```

Using operator `+` to concatenate strings, we should remember about some restrictions imposed by this type of characteristics. Type `str` in Python is an immutable type. (check: Which of the following types is mutable?).
Using operators `+` or `+=` with type `str` will create a new object `str`.
In the most cases use of `+` operator will be a good solution because of its understandable usage. Anyway, you must remember that when we combine a lot of strings of characters, it can have negative performance consequences. In this case it's better to use `join()` method.

Information about `str` type:
https://docs.python.org/3/library/stdtypes.html#str
https://docs.python.org/3/library/stdtypes.html#typesseq-common

Article about various string operations:
https://realpython.com/python-string-split-concatenate-join/

3. Which of the following instructions does not perform any operation?

Correct answer `pass`

`pass` statement does **not** do any operation. It can be used as a fulfillment, e.g. a function body, thus creating a structurally correct implementation:

```
def not_sure_about_it():
  pass
```

Information about `pass` statement in documentation:
https://docs.python.org/3/reference/simple_stmts.html#the-pass-statement

4. Which of the following keywords is used to define a conditional statement?

Correct answer is `if`

In Python conditions statement are written according to the following pattern :

```
if logical_condition:
  code that will be executed when the logical_condition is tru
e
```

After this section we can put any number of optional blocks `elif` to define alternative conditions:

```
elif another_condition:
  code that will be executed when none of the previous conditi
ons was true but this one is true
elif yet_another_condition:
  ...
```

At the very end we can define optional instruction `else` which will execute a code if none of all the previous conditions was true:

```
else:
 code that will be executed when none of all the previous cond
itions was true
```

In Python it is also possible to write instruction `if` in one line or use conditional expression (also called *ternary operator*).
More information:
https://realpython.com/python-conditional-statements/

5. How to correctly declare a function called `get_full_name`, having two arguments: `first_name` and `last_name`?

Correct answer is:

```
def get_full_name(first_name, last_name):
```

In Python the function can be declared using the keyword `def` next giving the name of the function (we can use this name to refer to the function to call it). Next, inside the round brackets we should type the names of the function arguments. They are optional (the function may have no arguments at all - then the brackets are empty inside: `()` .
The line ends with a colon, and the next line starts with the definition of the function body (instructions to be executed within it). The rule of thumb is not to pass too many arguments to the function. When a function receive a lot of arguments means that maybe its scope is too big and we should considerate dividing the function into several smaller ones.
Documentation:
https://docs.python.org/3/tutorial/controlflow.html#defining-functions

Basic information about functions in Python:
https://www.programiz.com/python-programming/function

Easy

1. What is the name of the first argument of the method (function in a class)?

Correct answer is `self`

The first argument the method will receive is a reference pointing to the instance of the class (object) on which the method was called. This argument is passed implicitly (when calling the method, we do not to pass it. Python will "add it" automatically running the function call).

Method example:

```
class Contract:
  def generate(self, law_policy):
    pass
```

and the method call:

```
from law_policy import polish_law


employee_contract = Contract()
employee_contract.generate(polish_law)
```

In this example, in the body of the `generate` method, using the argument `self` we can access the object written in `employee_contract` variable. In the `law_policy` argument, we will find the value passed as `polish_law`.
Naming the first argument of the method in a different way (e.g. `object_reference`) would be syntactically correct (it would work), however, it is inconsistent with the conventions and stylistic rules of Python code, defined by PEP 8: https://www.python.org/dev/peps/pep-0008/#function-and-method-arguments

2. What is the result of the action: `1_2 + 3_4` ?

Correct answer is 46.

Notation `1_2` is equivalent to `12` - it is just a number `12`. Underlines will be ignored by the interpreter. Similarly in the case of `3_4`. Underline notation was introduced in Python 3.6 to make it easier to write especially large numbers, e.g. instead of:

```
money = 10000000
```

we can write:

```
money = 10_000_000
```

which is easier to read.
This idea was introduced by PEP 515:
https://www.python.org/dev/peps/pep-0515/

3. Which of the following keywords is used to enable a context manager?

Correct answer is `with`.

Context manager allows to create readable and reusable imptementation of situation in which some instructions should always be executed before and after another dynamic part. A common example is working with files:

```
with open("path/to/file") as data_file:
  for line in data_file:
    print(line)
```

When working with various files we always want to:

- open the indicated file
- make specific operations on file
- close the file

What is important, the file should always be closed, regardless of whether the operations performed on it were successful or were aborted by an exception. The usage of context manager guarantees the above. An alternative syntax would be the usage of: `try…except…finally`, however, for example when working with files, this would be a less readable solution and duplicating the logic of error handling and file closing.

More information about context manager:
https://docs.python.org/3/reference/compound_stmts.html#with

4. Which of the following instructions is used to enable a csv module in our code?

Correct answer is
`import csv`

In Python, a module is just a file containing a Python code. To use a different module in the implemented solution (and available functions, classes, variables, etc.), it must be imported. This is the information for the interpreter that in the following code there will most likely be a reference to this module so the interpreter should find it. We should notice, that information about where the module is located and under what symbol it will be available, is passed in the same way. The import system should be used with the usage of:

- module from the standard library
- module from an external, previously installed library
- another module implemented within the same application (e.g. class or function from another file)

Modules are organized into hierarchical structures - like files on hard disk. This structure consists of modules called packages (which are equivalent to a directory on your hard disk), which may contain other packages (subdirectories) or modules that aren't packages (files). When importing a specific module, its location should be indicated using dotted notation to separate subsequent levels of depth:

```
>>> import os.path
>>> print(os.path.join("directory", "file"))
"directory/file"
```

To refer to an imported module, function etc. without dot notation, use construction: `from … import …`

```
>>> from os import path
>>> print(path.join("directory", "file"))
"directory/file"
```

It is also possible to give an alias to an imported module and refer to it via this alias:

```
>>> from os import path as system_path
>>> print(system_path.join("directory", "file"))
"directory/file"
```

The path to the module can be given both in absolute (preferred in most cases) and relative.

More details about modules import in Python:
https://docs.python.org/3/reference/import.html

PEP 8 (import):
https://www.python.org/dev/peps/pep-0008/#imports

Articles:
https://www.digitalocean.com/community/tutorials/how-to-import-modules-in-python-3
https://realpython.com/absolute-vs-relative-python-imports/

5. Which of the following instructions will return the length of the string: `"abc"` (number of characters)?

Correct answer is `len("abc")`

The built-in `len` function returns the number of elements (length) in a object. The type `str` is a sequence of characters and in this case the function `len` returns the length of the sequence (number of characters / letters). For the list, the function will return the number of elements in the list and for the dictionary the number of pairs: `key: value`. To enable the `len` function for your own class, you must meet the expected protocol. In this case, implement the `__len__` method. When someone uses the `len` function on an object of this class, the `__len__` method will be called and the returned value will be the result of the `len` function.

Information about `len` function in documentation:
https://docs.python.org/3/library/functions.html#len

Basic information about `__len__`:
https://stackoverflow.com/questions/2481421/difference-between-len-and-len

Medium:

1. The following code snippet: `some_variable = [number for number in range(10)]` is an example of usage of:

Correct answer: List comprehension
List comprehension is a short form of creating a new list based on an existing one (list or another sequence). This is an alternative to using a standard loop for.

```
some_variable = [number for number in range(10)]
```

Can be also written:

```
some_variable = []
for number in range(10):
  some_variable.append(number)
```

In this case, the use of comprehension lists allows to get a concise and easy to read form. In this example, the role of an existing sequence is `range(10)`, which is a sequence of numbers from 0 to 9 inclusive. List comprehension may also have a condition that allows you to filter the elements:

```
some_variable = [number for number in range(10) if number > 5]
```

In an extensive example:

```
class Student:
  def __init__(self, final_grade):
    self.final_grade = final_grade


students = [Student(3), Student(5), Student(6), Student(5)]
best_students = [student for student in students if student.fi
nal_grade > 4]
```

Similarly to creating nested loops, it is also possible to nest comprehension lists. However, you should keep in mind that for more complex expressions, it is better to use a loop approach, because complex list comprehensions become quickly very complicated.
In Python, there are also dict comprehensions, set comprehensions (working analogously to list comprehensions, but on other data types) and an expressions generator (working in a similar way).

More details:
https://docs.python.org/3/tutorial/datastructures.html#list-comprehensions

and more:
https://djangostars.com/blog/list-comprehensions-and-generator-expressions/

Information about range type:
https://docs.python.org/3.7/library/stdtypes.html#ranges

2. Which type is mutable?

Correct answer is `list`

In Python, as in most programming languages, data types can be mutable or immutable. Immutable objects can't be changed.
Type `str` is an example and operation of concatenation (merging, check: "Which of the following commands will assign the string " Python " to the `result` variable?"). To complete the name with the missing part, the `+=` operator can be used:

```
name = "Mik"
name += "ołaj"
```

But in this case, the original object representing the string "Mik" won't be modified, and a new object will be assigned to the name variable, created from the combination of "Mik" and "ołaj". This is illustrated in more detail:

```
>>> part_name = "Mik"
>>> name = part_name
>>> name is part_name
True
>>> name += "ołaj"
>>> name is part_name
False
>>> print(name)
Mikołaj
>>> print(part_name)
```

```
Mik
```

Therefore, modification of the immutable type is not possible. Each time a new object will be created and the "old" will remain intact. Among the built-in Python types are immutable::

- bool
- int
- float
- tuple
- frozenset
- str

On the other hand, a mutable object can be modified. An example of mutable types is:

- list
- set
- dict

A similar example to the previous one would look like this:

```
>>> numbers = [1, 2, 3]
>>> new_numbers = numbers
>>> new_numbers is numbers
True
>>> new_numbers.append(4)
>>> new_numbers is numbers
True
>>> print(new_numbers)
[1, 2, 3, 4]
>>> print(numbers)
[1, 2, 3, 4]
```

In this example, the list object is modified, and the change is visible in both, the "new" and "old" variables, because both point to the same object (no new object was created, as in the case of type `str`). This property has implications, when defining a function with a mutable argument with the default value.

More details:
https://docs.python.org/3/library/stdtypes.html#immutable-sequence-types
https://docs.python.org/3/glossary.html#term-immutable
https://docs.python.org/3/glossary.html#term-mutable

Article:
https://medium.com/@meghamohan/mutable-and-immutable-side-of-python-c2145cf72747

3. How to return the last element from the list: `grades = [5, 3, 2, 5]`?

Correct answer is `grades[-1]`

In Python we cane reference to individual items in the list by using the index, providing the item number in square brackets. The first item in the list has an index of 0. So calling:

```
print(grades[1])
```

will write `3`.

To start indexing from the end of the list we use negative values . So `-1` is the index of the last item in the list, `-2` the second to last, etc.

Sequence operations:
https://docs.python.org/3/library/stdtypes.html#common-sequence-operations

Information about lists:
https://www.programiz.com/python-programming/list

4. Which keyword is **NOT** the element of the block of exception handling (`try…`)

The correct answer is: `catch`

In Python, the exception handling block begins with the keyword `try:` followed by the code executing within the block. Next is the exception handling section (`except`, may occur several times) or the `finally` section, which will always be executed, regardless of whether an exception occurred. After the keyword `except`, you can specify the type of exception to be "caught" and handled in the block. If the thrown exception does not match the given type, the interpreter will look for the next `except` section with the matching type. Not specifying any type catches all types of exceptions. This approach is not recommended because it limits the flexibility of implementation. A good practice is to give the most accurate type of exceptions caught. To handle several different types of exceptions in a different way, prepare a separate `except`. section for each. In the exception handling block it is allowed to omit the `except` or `finally` section, however one of them must occur (if both, then in the order: first `except` then `finally`). Finally, there may be an optional `else` section if the exception is not thrown.

An example of an exception handling block might look like this:

```
data_file = open("path/to/file")
try:
  for line in data_file:
    print(line)
except IOError:
  print("Something went wrong...")
finally:
  data_file.close()
else:
  print("No problem :)")
```

The above example is simplified, for the purposes of presenting the possibilities of the `try…` structure. This applies primarily to error handling in the `except` block. Writing a message on the screen is not a good way to handle an exceptional cases. If all we should do with a given exception is to save information about its occurrence, one should use e.g. the logging module and store more detailed data in the program log. In terms of file support itself, the preferred solution is to use the available context manager (check: "What keyword can we use context manager for?").

Information about exceptions and their handling:
https://docs.python.org/3.7/tutorial/errors.html

5. How to declare a class `FullTimeContract`which inherits from the class `Contract`?

Correct answer is `class FullTimeContract(Contract):`

Inheritance means that the type (inheriting, child) is a more detailed version of the general type (base type, parent). In the example with the contract, `FullTimeContract` is a particular type of general `Contract`. Thus objects of different types inheriting from the same parent to be treated in the same way and to determine which class, the method being called, should be derived from when it is called. This concept is called polymorphism.

```
class Contract:
  def calculate_value(self, salary_per_hour):
    return salary_per_hour

class FullTimeContract(Contract):
  def calculate_value(self, salary_per_hour):
    return 160 * salary_per_hour

class PartTimeContract(Contract):
  def calculate_value(self, salary_per_hour):
    return 80 * salary_per_hour

contracts = [FullTimeContract(), PartTimeContract()
total_amount = 0
for contract in contracts:
  total_amount += contract.calculate_value(salary_per_hour=10
0)
```

In the implementation of a child class, you can refer to the properties and methods defined in the base class using the `super()` construct. The child class can also extend the base class by adding new fields and methods, as well as overwriting the implementation of a method already defined in the base class. Classes are associated with the object-oriented programming paradigm, which is a very broad topic, but also an important element of programming skills.

Information about classes in Python:
https://docs.python.org/3/tutorial/classes.html

Difficult:

1. Declaration `def get_distance() → Optional[int]:` means that the function `get_distance` should:

Correct answer is: Always return a value of type `int` or a value of `None`

The notation used above is type: hints. It allows to declare expected types for function arguments, returned values, variables. Python is a dynamically typed language - however, the introduction of bigger support for type hints in new versions of the language allows to use additional mechanisms that facilitate reading of existing code and to reduce the number of errors. The above statement won't prevent the function from being implemented in the following way:

```
def get_distance() -> Optional[int]:
  return "This is wrong"
```

but it will be easier to detect this type of error:

- Development environments (IDE like PyCharm) will indicate this implementation as a potential error
- Information about the expected types allows for better syntax suggestion
- Using the so-called type checker (e.g. mypy tool) error will be returned when the code is checked (e.g. before joining changes to the common repository), and not when it is executed in a test or production environment

The entry `→ int` means that the given function should always return a value of type `int`. "Packaging" in `Optional` also allows to return value `None`.

Information about the type hints:
https://docs.python.org/3/library/typing.html

Mypy tool:
https://github.com/python/mypy

Carl Meyer about type hints:
https://youtu.be/pMgmKJyWKn8

2. What type can NOT be a key in a dictionary?

Correct answer is `list`

The key in the dictionary can be any object that is "hashable". This means objects for which the hash value doesn't change and can be compared with others based on their identity, not the value. Both `int`, `str` and `bool` are correct dictionary keys. The list, like other built-in mutable types, isn't hashable, and therefore can't be used as a key in the dictionary.

Information about dictionary and hashable:
https://docs.python.org/3/library/stdtypes.html#typesmapping
https://docs.python.org/3/glossary.html#term-hashable

a. How to correctly use an empty list as a default argument for a function?

The correct answer is:

```
def calculate(numbers=None):
  if numbers is None:
    numbers = []
```

This construction is due to the fact that the list is a mutable type and to the way Python processes the function definition and default arguments. Default values are assigned to function arguments when the interpreter "reads" its definition for the first time, not after each function call. So this only happens once during the whole program execution. Giving an empty list directly as the default argument will cause it to be one and the same list for each call. Because this is a mutable type, all changes made on the first call will be visible on the second. The following example illustrates this well:

```
def more_numbers(numbers=[]):
  print(numbers)
  numbers.append(1)


>>> more_numbers()
[]
>>> more_numbers()
[1]
>>> more_numbers()
[1, 1]
```

In each subsequent call, the function uses the same list that has already been modified by previous executions. The use of the structure using `None` as the default value and assigning the list to a variable only in the body of the function is an accepted way to solve this problem. A similar situation applies to all mutable types being used as the default arguments of the function.

W każdym kolejnym wywołaniu funkcja korzysta z tej samej listy, która została już wcześniej zmodyfikowana przez poprzednie wykonania. Zastosowanie konstrukcji z wykorzystaniem `None` jako domyślnej wartości i przypisanie listy do zmiennej dopiero w ciele funkcji jest przyjętym sposobem na rozwiązanie tego problemu. Analogiczna sytuacja dotyczy wykorzystywania wszystkich typów mutable jako domyślnych argumentów funkcji.

Default arguments:
https://docs.python.org/3/tutorial/controlflow.html#default-argument-values

Using mutable types as the default arguments of the function:
https://nikos7am.com/posts/mutable-default-arguments/

4. Which of the function `def download(url, timeout=5):` invocation is NOT correct?

Correct answer is `download(url="www.infoshareacademy.com", 10)`

Such a call will cause an error (SyntaxError) resulting from passing a positional argument after the named argument. Arguments passed to a function or method can be positional arguments (without the argument name and the `=` character) or named arguments (keyword). For a function defined in this example, it's possible to:

- pass both arguments as positional arguments `download("www.infoshareacademy.com", 10)`
- pass only the first argument as a positional argument (timeout will default) `download("www.infoshareacademy.com")`
- pass only the first argument as a keyword argument (timeout will default) `download(url="www.infoshareacademy.com")`
- pass both arguments as keyword arguments `download(url="www.infoshareacademy.com", timeout=10)` or `download(timeout=10, url="www.infoshareacademy.com")`

The order of keyword arguments isn't important (the interpreter knows which value should be assigned to which argument). The order of positional arguments is important because it corresponds to their assignment to individual variables. It is not allowed to pass the positional argument after keyword arguments, or is it allowed to omit an argument that has no default value.

Introduced in PEP 3102 concept of the Keyword-Only Arguments allows you to force some (or all) arguments in the form of keyword arguments. This is to increase readability in situations where the value of the argument isn't obvious in the context of the function being called. To apply this convention, you must use an asterisk when you declare function arguments. This will cause all subsequent arguments to be passed as keywords arguments:

```
def something(could_be_positional, *, only_keyword):
  pass
```

Information about arguments of the function:
https://docs.python.org/3/tutorial/controlflow.html#more-on-defining-functions

PEP 3102:
https://www.python.org/dev/peps/pep-3102/

5. How to return a list, having elements indexed 1, 2 and 3 from the list `grades = [5, 3, 2, 5, 6]`?

Correct answer is `grades[1:4]`

By referencing the list (or other sequence) using an index and a colon, you can perform the slice operation. This operation allows you to receive:

- list of items from the beginning to the given index `[:index]` (with no element indexed `index`)
- list of items from the given index to the end `[index:]` (with element indexed `index`)
- list of items between two indexes `[start:end]` (with element on index `start`, without element on index `end`)
- list of elements between two indexes, containing every n-th element `[start:end:n]` (with item on index `start`, then the element at index `start + n` etc., without indexed element `end`)

Slice operations are a convenient way to get a selected subset of elements from a longer sequence. When using a slice, remember:

- the first element of the list has the index 0
- last element of the list assigned to the variable `numbers` ma indeks `-1` albo `len(numbers) - 1`
- the use of complicated slices (e.g. using negative indexes and selecting every n-th element) significantly reduces the clearness of the code. In such case, it is worth carrying out the planned operation e.g. in several steps

Lists and list operations:
https://docs.python.org/3/tutorial/introduction.html#lists
https://docs.python.org/3/library/stdtypes.html#common-sequence-operations