



# pythonprogramming

Assoc. Prof. Dr. Bora Canbula



[github.com/canbula/PythonProgramming](https://github.com/canbula/PythonProgramming)

# Variables

Variables are symbols for memory addresses.

## Built-in Functions

The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order.

### Built-in Functions

#### A

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

#### E

`enumerate()`  
`eval()`  
`exec()`

#### L

`len()`  
`list()`  
`locals()`

#### R

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

#### F

`filter()`

#### M

`map()`

#### S

### **hex(x)**

Convert an integer number to a lowercase hexadecimal string prefixed with "0x". If *x* is not a Python `int` object, it has to define an `__index__()` method that returns an integer. Some examples:

```
>>> hex(255)
'0xff'
>>> hex(-42)
'-0x2a'
```

```
>>>
```

`classmethod()`  
`compile()`  
`complex()`

`help()`  
**hex()**

`ord()`

`type()`

#### P

`pow()`  
`print()`

#### V

`vars()`

#### D

**id()**

### **id(object)**

Return the "identity" of an object. This is an integer which is guaranteed to be unique and constant for this object during its lifetime. Two objects with non-overlapping lifetimes may have the same `id()` value.

# Identifier Names

For variables, functions, classes etc. we use identifier names. We must obey some rules and we should follow some naming conventions.

## Rules

- ▶ Names are case sensitive.
- ▶ Names can be a combination of letters, digits, and underscore.
- ▶ Names can only start with a letter or underscore, can not start with a digit.
- ▶ Keywords can not be used as a name.



## keyword — Testing for Python keywords

Source code: [Lib/keyword.py](#)

This module allows a Python program to determine if a string is a **keyword** or **soft keyword**.

**keyword.iskeyword(s)**

Return **True** if *s* is a Python **keyword**.

**keyword.kwlist**

Sequence containing all the **keywords** defined for the interpreter. If any keywords are defined to only be active when particular **\_\_future\_\_** statements are in effect, these will be included as well.

**keyword.issoftkeyword(s)**

Return **True** if *s* is a Python **soft keyword**.

*New in version 3.9.*

**keyword.softkwlist**

Sequence containing all the **soft keywords** defined for the interpreter. If any soft keywords are defined to only be active when particular **\_\_future\_\_** statements are in effect, these will be included as well.

*New in version 3.9.*

# Identifier Names

For variables, functions, classes etc. we use identifier names. We must obey some rules and we should follow some naming conventions.

## Rules

- ▶ Names are case sensitive.
- ▶ Names can be a combination of letters, digits, and underscore.
- ▶ Names can only start with a letter or underscore, can not start with a digit.
- ▶ Keywords can not be used as a name.

<https://peps.python.org/>

Python Enhancement Proposals [Python](#) » [PEP Index](#) » PEP 8



## PEP 8 – Style Guide for Python Code

**Author:** Guido van Rossum <guido at python.org>, Barry Warsaw <barry at python.org>, Nick Coghlan <ncoghlan at gmail.com>

**Status:** Active

**Type:** Process

**Created:** 05-Jul-2001

**Post-History:** 05-Jul-2001, 01-Aug-2013

# Identifier Names

For variables, functions, classes etc. we use identifier names. We must obey some rules and we should follow some naming conventions.

## Conventions

- ▶ Names to Avoid  
Never use the characters 'l' (lowercase letter el), 'O' (uppercase letter oh), or 'I' (uppercase letter eye) as single character variable names.
- ▶ Packages  
Short, all-lowercase names without underscores
- ▶ Modules  
Short, all-lowercase names, can have underscores
- ▶ Classes  
CapWords (upper camel case) convention
- ▶ Functions  
snake\_case convention
- ▶ Variables  
snake\_case convention
- ▶ Constants  
ALL\_UPPERCASE, words separated by underscores

## Leading and Trailing Underscores

- ▶ \_single\_leading\_underscore  
Weak “internal use” indicator.  
`from M import *` does not import objects whose names start with an underscore.
- ▶ single\_trailing\_underscore\_  
Used by convention to avoid conflicts with keyword.
- ▶ \_\_double\_leading\_underscore  
When naming a class attribute, invokes name mangling (inside class `FooBar`, `__boo` becomes `_FooBar__boo`)
- ▶ \_\_double\_leading\_and\_trailing\_underscore\_\_  
“magic” objects or attributes that live in user-controlled namespaces (`__init__`, `__import__`, etc.). Never invent such names; only use them as documented.

# Variable Types

Python is dynamically typed. Python does not have primitive types. Everything is an object in Python, therefore, a variable is purely a reference to an object with the specified value.

## Numeric Types

- ▶ Integer
- ▶ Float
- ▶ Complex
- ▶ Boolean

## Sequences

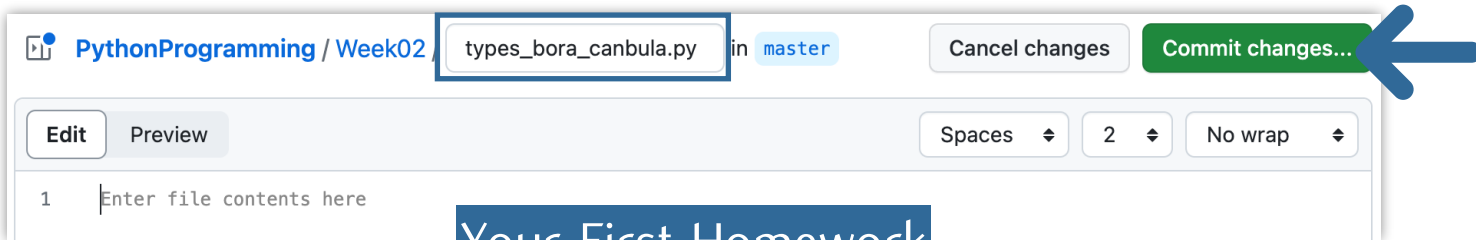
- ▶ Strings
- ▶ List
- ▶ Tuple
- ▶ Set
- ▶ Dictionary

## Your First Homework

The screenshot shows the GitHub interface for the 'PythonProgramming' repository. A blue arrow points to the 'Week02' folder in the file list. Another blue arrow points to the 'Add file' button in the top right of the repository view. Below the repository view, a message box states: 'You need to fork this repository to propose changes. Sorry, you're not able to edit this repository directly—you need to fork it and propose your changes from there instead.' A green button labeled 'Fork this repository' and a blue link 'Learn more about forks' are provided.

**You need to fork this repository to propose changes.**  
Sorry, you're not able to edit this repository directly—you need to fork it and propose your changes from there instead.

[Fork this repository](#)  
[Learn more about forks](#)



## Your First Homework

- ☒ An integer with the name:  
`my_int`
- ☒ A float with the name:  
`my_float`
- ☒ A boolean with the name:  
`my_bool`
- ☒ A complex with the name:  
`my_complex`

## Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#) or [learn more about diff comparisons](#).

base repository: canbula/PythonProgramming    base: master    ↩

head repository: JabbaBC/PythonProgramming    compare: patch-1    ↩

✓ **Able to merge.** These branches can be automatically merged.

Discuss and others. [Learn](#)

**Create pull request**

**Add a title**

Create types\_bora\_canbula.py

**Add a description**

Write    Preview    H    B    I    ≡    <>    ↻    ≡    ≡    ≡    ...

## Describe your changes

## Checklist

- [ ] I have read the [CONTRIBUTING]
- [ ] I have performed a self-review of my own code
- [ ] I have run the code locally and it works as expected
- [ ] I have commented my code, particularly in hard-to-understand areas

## Screenshots (if appropriate)

<!-- Add screenshots here if appropriate -->

Markdown is supported    Paste, drop, or click to add files

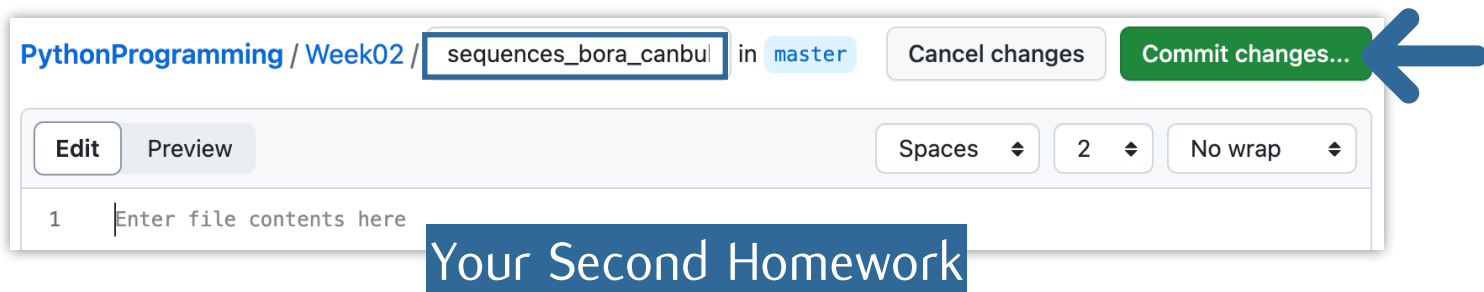
☒ Allow edits by maintainers    **Create pull request**

**All checks have failed**    [Hide all checks](#)

1 failing check

- Python application / build (pull\_request)    Failing after 18s    [Details](#)
- This branch has no conflicts with the base branch

Only those with [write access](#) to this repository can merge pull requests.



- ✓ A list with the name:  
`my_list`
- ✓ A tuple with the name:  
`my_tuple`
- ✓ A set with the name:  
`my_set`
- ✓ A dictionary with the name:  
`my_dict`
- ✓ A function with the name:  
`remove_duplicates (list -> list)`  
to remove duplicate items from a list
- ✓ A function with the name:  
`list_counts (list -> dict)`  
to count the occurrence of each item  
in a list and return as a dictionary
- ✓ A function with the name:  
`reverse_dict (dict -> dict)`  
to reverse a dictionary, switch values  
and keys with each other.



# Problem Set

1. What is the correct writing of the programming language that we used in this course?

- ☐ ( ) Phyton
- ☐ ( ) Pyhton
- ☐ ( ) Pthyon
- ☐ ( ) Python

2. What is the output of the code below?

```
my_name = "Bora Canbula"  
print(my_name[2::-1])
```

- ☐ ( ) alu
- ☐ ( ) ula
- ☐ ( ) roB
- ☐ ( ) Bor

3. Which one is not a valid variable name?

- ☐ ( ) for\_
- ☐ ( ) Manisa\_Celal\_Bayar\_University
- ☐ ( ) IF
- ☐ ( ) not

4. What is the output of the code below?

```
for i in range(1, 5):  
    print(f"{i:2d} {(i/2):4.2f}", end='')
```

- ☐ ( ) 010.50021.00031.50042.00
- ☐ ( ) 10.50 21.00 31.50 42.00
- ☐ ( ) 1 0.5 2 1.0 3 1.5 4 2.0
- ☐ ( ) 100.5 201.0 301.5 402.0

5. Which one is the correct way to print Bora's age?

```
profs = [  
    {"name": "Yener", "age": 25},  
    {"name": "Bora", "age": 37},  
    {"name": "Ali", "age": 42}  
]
```

- ☐ ( ) profs["Bora"]["age"]
- ☐ ( ) profs[1][1]
- ☐ ( ) profs[1]["age"]
- ☐ ( ) profs.age[name="Bora"]

6. What is the output of the code below?

```
x = set([int(i/2) for i in range(8)])  
print(x)
```

- ☐ ( ) {0, 1, 2, 3, 4, 5, 6, 7}
- ☐ ( ) {0, 1, 2, 3}
- ☐ ( ) {0, 0, 1, 1, 2, 2, 3, 3}
- ☐ ( ) {0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4}

7. What is the output of the code below?

```
x = set(i for i in range(0, 4, 2))  
y = set(i for i in range(1, 5, 2))  
print(x^y)
```

- ☐ ( ) {0, 1, 2, 3}
- ☐ ( ) {}
- ☐ ( ) {0, 8}
- ☐ ( ) SyntaxError: invalid syntax

8. Which of the following sequences is immutable?

- ☐ ( ) List
- ☐ ( ) Set
- ☐ ( ) Dictionary
- ☐ ( ) String

9. What is the output of the code below?

```
print(int(2_999_999.999))
```

- ☐ ( ) 2
- ☐ ( ) 3000000
- ☐ ( ) ValueError: invalid literal
- ☐ ( ) 2999999

10. What is the output of the code below?

```
x = (1, 5, 1)  
print(x, type(x))
```

- ☐ ( ) [1, 2, 3, 4] <class 'list'>
- ☐ ( ) (1, 5, 1) <class 'range'>
- ☐ ( ) (1, 5, 1) <class 'tuple'>
- ☐ ( ) (1, 2, 3, 4) <class 'set'>