



Python Basics



Session-2





3

What is “Docstrings” ?



Students, write your response!



Docstrings

- ▶ **Docstrings** are - unlike regular comments - stored as an attribute of the *function* or the *module* they document, meaning that you can access them programmatically.
- ▶ Docstring runs as **an explanatory text** of codes and it should be written between triple quotes.

```
"""...docstring text..."""
```

Docstrings



A **symbolic** representation of the docstring.

```
def function_name():  
    """
```

```
    Hi, I am the docstring of this code.  
    If you need any information about this function or module, read me.  
    It can help you understand how the module or function works.  
    """
```

```
    .  
    .  
    .  
    .  
    .
```

The code block
of the function

Docstring of the
function

Docstrings

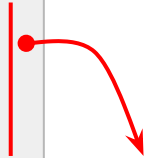


A **symbolic** representation of the docstring.

```
toaster :  
""  
Hi, This toaster works with 110v electricity.  
You can use it to toast bread and bagels.  
Please use for less than 30 mins continuous.  
Fire Hazard.  
Created by Clarusway.  
""
```



The code block
of the function



Docstring of the
function



Docstrings

► Displaying the docstring of a function :

```
print(function_name.__doc__)
```

Normally, when we want to call docstring of a function or module to read, we will use `.__doc__` (the keyword `doc` enclosed by double underscores) syntax.



Docstrings

► Displaying the docstring of a function :

```
print(function_name.__doc__)
```

Normally, when we want to call docstring of a function or module to read, we will use `.__doc__` (the keyword `doc` enclosed by double underscores) syntax.

```
Hi, I am the docstring of this code.  
If you need any information about this function or  
module, read me.  
It can help you understand how the module or function  
works.
```



Docstrings

► Displaying the docstring of the `print()` function :

input :

```
1 print(print.__doc__)  
2
```

output :

```
1 print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)  
2  
3 Prints the values to a stream, or to sys.stdout by default.  
4 Optional keyword arguments:  
5 file:  a file-like object (stream); defaults to the current sys.stdout.  
6 sep:   string inserted between values, default a space.  
7 end:   string appended after the last value, default a newline.  
8 flush: whether to forcibly flush the stream.  
9
```




Docstrings

- **Display the docstring of the following several built-in functions on your Playground.** You don't have to know/learn what these functions used for, for now.. :

```
map()    : print(map.__doc__)  
sum()    : ...  
input(): ...
```



Naming Variables



Table of Contents



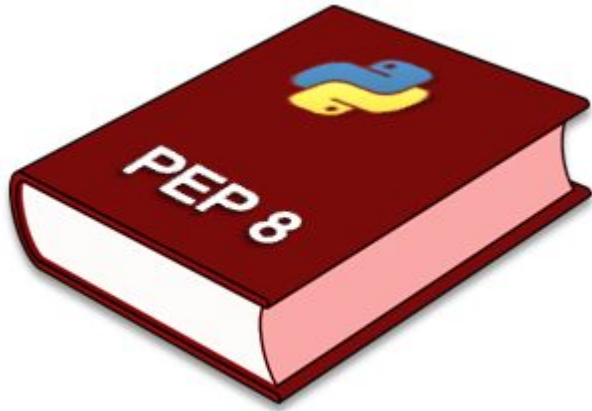
- ▶ General Description
- ▶ Conventional (PEP 8) Naming Rules



1

General Description

General Description



- ▶ Expert programmers care much for naming the variables well to make their codes easy to understand.
- ▶ It is important because programmers spend a lot of time reading and understanding code written by other programmers.

General Description



Tips:

- Remember, a nice and meaningful naming of variables is a skill that can be gained over time. Of course, you also need to be familiar with PEP 8 traditional rules.

Of course, the conventional rules of naming is **optional**. You can use any names you like but it is useful to follow these rules so that someone (**including you**) knows what you have written.



I didn't understand anything. **What can it be?**

2 Con_PEP_N_RL



Students, write your response!



Con_PEP_N_Rl



bad example of naming
a variable



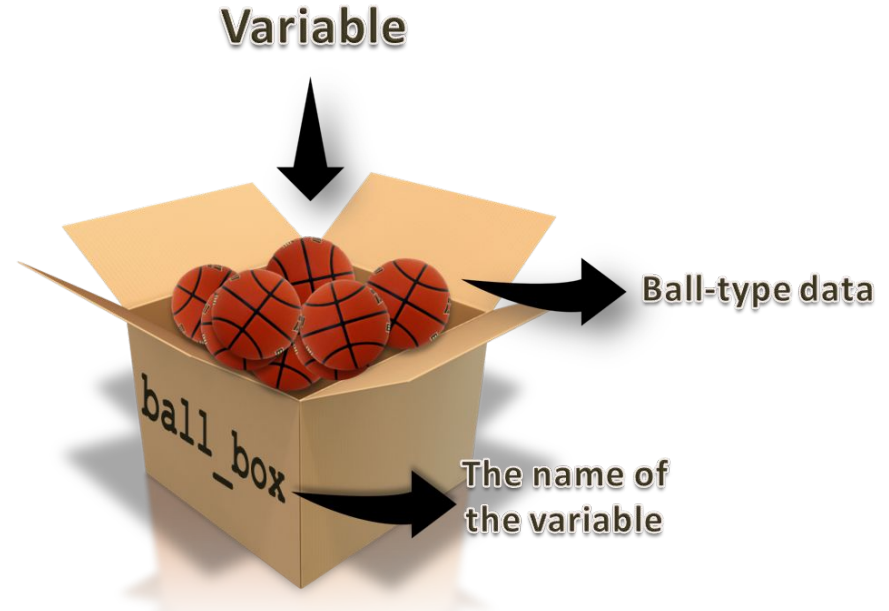
2

Conventional (PEP 8)
Naming Rules

Variables




- ▶ Variable is a location designated where a value can be stored and accessed later. Imagine a box where you store something. That's a variable.



Variables



Creating a variable is very simple in Python.

All you need to do is specify the variable name and then assign a value to it using  **=**

variable name **=** value

```
planet = 'jupyter'  
price = 140  
pi_number = 3.14
```

**The declaration
happens automatically
when you **assign** a
value to a variable.**



Variables

► Task

- ▶ Create 3 variables and assign different values to them.
- ▶ Display each of them in Python Playground using `print()` function.



Variables



```
my_age = 33  
your_age = 30  
my_age = your_age  
print(my_age)
```

What is the output? Try to figure out in your mind...





Conventional (PEP 8) Naming Rules

► Some basic naming conventions

- Choose **lowercase** words and use underscore to split the words

```
variable = 3.14
```



Conventional (PEP 8) Naming Rules

► Some basic naming conventions

- Do not use `'l'` (**lowercase letter “L”**) as single character variable.

```
l = 3.14  # This is lowercase letter el  
I = 3.14  # This is uppercase letter eye
```



Conventional (PEP 8) Naming Rules

► Some basic naming conventions

- Do not use `'0'` (**uppercase letter “O”**) as single character variable.

```
time_0 = '3.14'    # This is uppercase letter “O”  
time_0 = '3.14'    # This is number zero
```



Some Important PEP 8 Rules

- ▶ **Some basic naming conventions (reserved words)**

- ▶ Do not use specific Python keywords such as :

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	break
except	in	raise		

Pythonic Rules



Examine these samples carefully

2me



data4me



Big boss



first!



big-boss



xy#@!v



last_name



\$price





Some Important PEP 8 Rules

► Some basic naming conventions

- The name of the variable must be legible, meaningful and relevant to the type of value

Good

```
students = ...
```

Bad

```
s = ... or st = ...
```



Some Important PEP 8 Rules

► Some basic naming conventions

- The name of the variable must be legible, meaningful and relevant to the type of value

Good

```
students = ...
```

```
# Big Data
```

```
big_data = ...
```

Bad

```
s = ... or st = ...
```

```
# Big Data
```

```
b_dt = ...
```



Some Important PEP 8 Rules

► Some basic naming conventions

- The name of the variable must be legible, meaningful and relevant to the type of value

Good

```
students = ...
```

```
# Big Data
```

```
big_data = ...
```

```
# Average income of February
```

```
avg_income_feb = ...
```

Bad

```
s = ... or st = ...
```

```
# Big Data
```

```
b_dt = ...
```

```
# Average income of February
```

```
average_income_february = ..
```



► Naming Variable

amount of rotten fruits

the list of prime numbers

the list of mathematics exam scores

What can be the **Name** of these sentences as variables.





Naming variable

```
amount of rotten fruits
```

- **Good samples :**

- rotten_fruit_amount = 33 # kg.
- amnt_fruit_rotten = 33 # kg.
- amount_rotten_fruits = 33 # kg.



Naming variable

the list of prime numbers

- **Good samples :**
 - prime_list
 - prime_no
 - list_prime
 - num_prime



► Naming variable

the list of mathematics exam scores

- **Good samples :**
 - math_scores
 - score_maths

Variables



Assigning a value to a variable.

```
x = y = z = "same"
```

```
print(x)  
print(y)  
print(z)
```

What is the output?





Variables



The output

same
same
same

Write a Python code on Playground :

Hint : Use int value of 30, 31, 28 only once.

Which months have **31** days and which have **30** or **28**? Let's assign the number of days (30 or 31 or 28) to the months (the variables will be the name of the months) in totally **three** code **lines** then print their number of days in order of the months as follows.

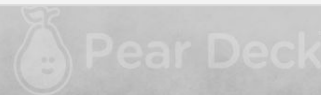


```
print(january, february, march, april, may, june, july, august, september, october, november, december)
```

31 28 31 30 31 30 31 31 30 31 30 31



Students, write your response!



Pear Deck Interactive Slide
Do not remove this bar

Variables



A probable answer :

```
january = march = may = july = august = october = december = 31
```

```
# multi assignments in a single line
```

```
april = june = september = november = 30
```

```
february = 28
```

```
print(january, february, march, april, may, june, july, august, september, october,  
november, december)
```

If we don't know the value of a variable, what can we assign to it? For example :

The ages of instructors :

```
thomas = 33
```

```
marry = 28
```

```
volter = ?
```

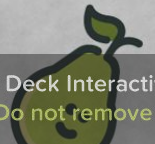
```
isabella = 46
```



Students, write your response!



Pear Deck Interactive Slide
Do not remove this bar





Variables



If we don't know the value of a variable, we can assign `None` to it.

```
thomas = 33  
marry = 28  
volter = None  
isabella = 46
```



Variables



Assigning a value to a variable.

```
website = "apple.com"  
print(website)
```

```
# assigning a new variable to website  
website = "clarusway.com"
```

```
print(website)
```

What is the output?





Variables



The output

apple.com
clarusway.com



Variables



Assigning a value to a variable.

```
first_number = 100  
second_number = first_number  
print(second_number)
```

What is the output?





Variables



The output

100

Variables



Assigning a value to a variable.

```
x = 15
```

```
y = 33
```

```
z = x
```

```
x = y
```

```
print(x)
```

```
print(y)
```

```
print(z)
```

What is the output?





Variables



The output

```
x = 33
```

```
y = 33
```

```
z = 15
```



Variables



Assigning a value to a variable.

```
a, b, c = 5, 3.2, "Hello"
```

```
print(a)
```

```
print(b)
```

```
print(c)
```

What is the output?





Variables



The output

5

3.2

Hello



Variables



Pay attention to the value of variables and how they change.

input :

```
1 color = 'red' # str type variable
2 season = 'summer'
3 price = 250 # int type variable
4 pi = 3.14 # float type variable
5 color = 'blue' # You can always assign a new value to a created variable
6 price = 100 # value of 'price' is changed
7 season = 'winter' |
8
9 print(color, price, season, sep=', ')
10
```

output :

```
1 blue, 100, winter
2
```



Variables

- Pay attention to the value of variables and how they change.

```
man = "andrew"  
color = "green"  
age = 32  
pi = 3.14  
color = "yellow"  
age = 44  
man = "joseph"  
  
print(man, age, color)
```

Output

```
joseph 44 yellow
```




Basic Data Types



Table of Contents



- ▶ Introduction to Data Types
- ▶ Strings
- ▶ Numeric Types
- ▶ Boolean
- ▶ Type Conversion



1

str

Introduction to Data Types

int

bool



► Introduction to Data Types

- Each data has a type.
- This type of data defines how you store it in memory and it also describes which process can be applied to it.



Introduction to Data Types



- ▶ **Some simple data types commonly used in Python.**
 - ▶ String,
 - ▶ Integer,
 - ▶ Float,
 - ▶ Boolean.





str

"2020"

3

Strings

"string is the most used type"

"i have 3 lb. of apple"



Strings



- ▶ If you want to work with any **textual** characters in your code, you have to work with strings.

```
my_text = 'being a good person'  
print(my_text)
```

String type is
called **str**.



```
type(variable)
```





Strings

- ▶ If you want to work with any **textual** characters in your code, you have to work with strings.

```
my_text = 'being a good person'  
print(my_text)
```

```
being a good person
```

String type is
called **str**.

- ▶ Strings are identified as a set of characters represented in the **single** or **double** quotes.

`type(variable)`

Type the following values on your Playground to print them as **str** type...

- `alfred@clarusway.com`
- `632`
- `It's early`

If you finish early, work with a peer and help them finish the activity..





Strings

- ▶ **Let's** do some practices which cover string type.

```
1 str_number = '1923'  
2 str_sign = '%(#&*?-'  
3  
4  
5 print(str_number)  
6 print(str_sign)  
7 print(type(str_number), type(str_sign))  
8  
9
```



Strings

- ▶ **Let's** do some practices which cover string type.

```
1 str_number = '1923'  
2 str_sign = '%(#&*?-'  
3  
4  
5 print(str_number)  
6 print(str_sign)  
7 print(type(str_number), type(str_sign))  
8  
9
```

Output

```
1923  
%(#&*?-  
<class 'str'> <class 'str'>
```



4

int

Numeric Types

float



Numeric Types

- ▶ Three basic numeric types in Python :

- Integers
- Floats
- Complexes



Numeric Types

- **Signed integer** types are whole numbers which don't contain decimal point.

```
my_integer = 40
negative_num = -18

print(my_integer)
print(negative_num)
```

Signed integer
type is called
int.



Numeric Types

- **Signed integer** types are whole numbers which don't contain decimal point.

```
my_integer = 40
negative_num = -18

print(my_integer)
print(negative_num)
```

```
40
-18
```

Signed integer
type is called
int.



Numeric Types

- **Floating point** types stand for real numbers with a decimal point.

```
my_float = 40.0  
negative_float = -18.66  
  
print(my_float)  
print(negative_float)
```

Floating point
type is called
float.



Numeric Types

- **Floating point** types stand for real numbers with a decimal point.

```
my_float = 40.0
negative_float = -18.66

print(my_float)
print(negative_float)
```

```
40.0
-18.66
```

Floating point
type is called
float.



5

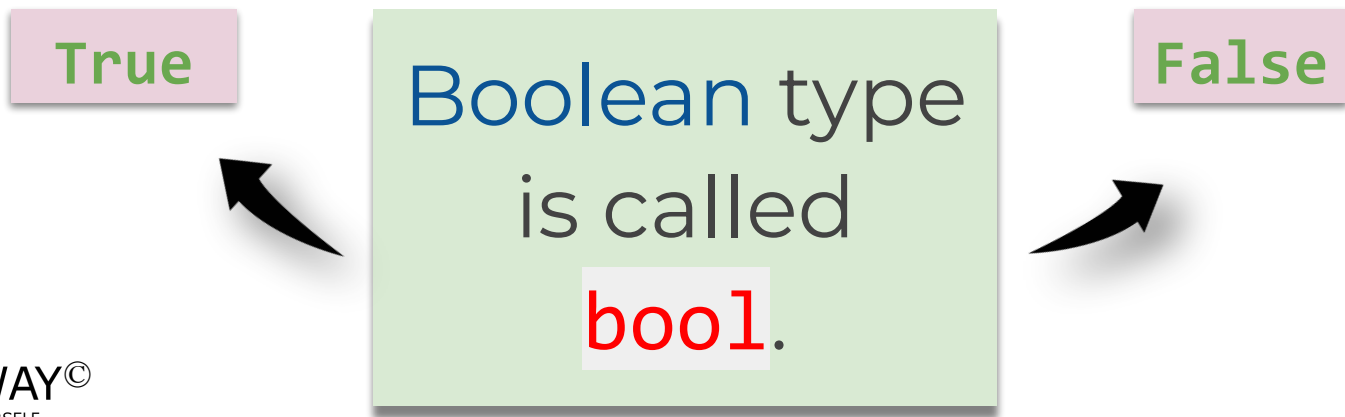
Boolean





Boolean

- ▶ Boolean types' values are the two constant objects **False** and **True**.
- ▶ In numeric contexts (for example, when used as the argument to an arithmetic operator), they behave like the integers 0 and 1, respectively.





6 Type Conversion

int ↔ str



Type Conversion



We can print the types of data using `type()` function

```
my_data = 'I am string'  
print(type(my_data))
```



Type Conversion



We can print the types of data using `type()` function

```
my_data = 'I am string'  
print(type(my_data))
```

```
<class 'str'>
```



Type Conversion



Type conversion functions.

We can convert the types of data to each other if the type allows to be converted. There are some functions to convert the types:

- `str()` converts to **string** type
- `int()` converts to **signed integer** type
- `float()` converts to **floating point** type

The value of any type in Python can be converted to a **str**.



Students choose an option

Pear Deck Interactive Slide
Do not remove this bar



Type Conversion



Converting **float** to **str**

```
pi = 3.14  
  
converted_pi = str(pi)  
print(converted_pi)  
print(type(converted_pi))
```



Type Conversion



Converting **float** to **str**

```
pi = 3.14  
  
converted_pi = str(pi)  
print(converted_pi)  
print(type(converted_pi))
```

```
3.14  
<class 'str'>
```

Type Conversion



Converting `float` to `int`

```
pi = 3.14  
  
converted_pi = int(pi)  
print(converted_pi)  
print(type(converted_pi))
```

What is the
output?



Students, write your response!

Type Conversion



Converting float to int

```
pi = 3.14  
  
converted_pi = int(pi)  
print(converted_pi)  
print(type(converted_pi))
```

```
3  
<class 'int'>
```

Type Conversion



Converting `int` to `float`

```
no = 3
```

```
converted_no = float(no)
```

```
print(converted_no)
```

```
print(type(converted_no))
```



Type Conversion



Converting `int` to `float`

```
no = 3  
  
converted_no = float(no)  
print(converted_no)  
print(type(converted_no))
```

```
3.0  
<class 'float'>
```



Type Conversion



input :

```
1 x = 39
2 v = "11"
3 y = "2.5"
4 z = "I am at_"
5
6 print(x-int(v))
7 print(x-float(y))
8 print(z+str(x))
9
```

output :

```
1 28
2 36.5
3 I am at_39
4
5
```



Type Conversion

input :

```
1 x = 39
2 v = "11"
3 y = "2.5"
4 z = "I am at_"
5
6 print(x-int(v))
7 print(x-float(y))
8 print(z+str(x))
9
```

$x - \text{int}("11") = 39 - 11 = 28$

output :

```
1 28
2 36.5
3 I am at_39
4
5
```




Type Conversion

input :

```
1 x = 39
2 v = "11"
3 y = "2.5"
4 z = "I am at_"
5
6 print(x-int(v))
7 print(x-float(y))
8 print(z+str(x))
9
```

$x - \text{int}("11") = 39 - 11 = 28$

$x - \text{float}("2.5") = 39 - 2.5 = 36.5$

output :

```
1 28
2 36.5
3 I am at_39
4
5
```



Type Conversion

input :

```
1 x = 39
2 v = "11"
3 y = "2.5"
4 z = "I am at_"
5
6 print(x-int(v))
7 print(x-float(y))
8 print(z+str(x))
9
```

$x - \text{int}("11") = 39 - 11 = 28$

$x - \text{float}("2.5") = 39 - 2.5 = 36.5$

$z + \text{str}(39) = \text{"I am at_"} + \text{"39"} = \text{I am at_39}$

output :

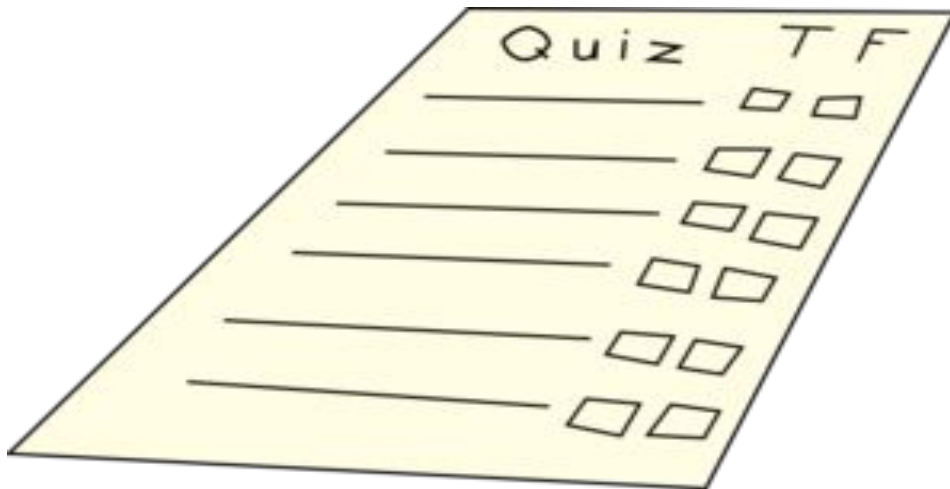
```
1 28
2 36.5
3 I am at_39
4
5
```

Type Conversion



► Task

- First, Login to your LMS,
- Then, click [here](#) to complete and submit the task.





Type Conversion



Without using any Interpreter/IDLE, just try to guess the output.

```
number_int = 123  
numberflt = 1.23
```

```
number_new = number_int + numberflt
```

```
print("datatype of number_int:", type(number_int))
```

```
print("datatype of numberflt:", type(numberflt))
```

```
print("Value of number_new:", number_new)
```

```
print("datatype of number_new:", type(number_new))
```

What is the output?





Type Conversion



The output

```
datatype of number_int: <class 'int'>  
datatype of number_flo: <class 'float'>  
Value of number_new: 124.23  
datatype of number_new: <class 'float'>
```



Type Conversion



Without using any Interpreter/IDLE, just try to guess the output.

```
number_int = 123  
number_str = "456"
```

```
print("Data type of number_int:", type(number_int))  
print("Data type of number_str:", type(number_str))  
  
print(number_int + number_str)
```

What is the output?





Type Conversion



The output

```
Data type of number_int: <class 'int'>
```

```
Data type of number_str: <class 'str'>
```

```
Traceback (most recent call last):
```

```
File "python", line 7, in <module>
```

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```



Type Conversion



Without using any Interpreter/IDLE, just try to guess the output.

```
number_int = 123  
number_str = "456"
```

What is the output?

```
print("Data type of number_int:", type(number_int))  
print("Data type of number_str before Type Casting:", type(number_str))  
  
number_str = int(number_str)  
print("Data type of number_str after Type Casting:", type(number_str))  
  
number_sum = number_int + number_str  
  
print("Sum of number_int and number_str:", number_sum)  
print("Data type of the sum:", type(number_sum))
```





Type Conversion



The output

```
Data type of number_int: <class 'int'>
```

```
Data type of number_str before Type Casting: <class 'str'>
```

```
Data type of number_str after Type Casting: <class 'int'>
```

```
Sum of number_int and number_str: 579
```

```
Data type of the sum: <class 'int'>
```



THANKS!

End of the Lesson (Basic Data Types)

next Lesson



Simple Operations

click above

