<PyNOOB>Explains:

Python: Print () function

____

print() : print function is a function used to display something (text, number...) on the screen.

How to use it?

I - Simple uses:

1) print("any text") *(You can use single quotes too)*

```
>>> print("Hello OjjOj !")
Hello OjjOj!
```

2) print(variable)

```
>>> a = "OjjOj"
>>> print(a)
OjjOj
```

⚠

If the variable was a number!

Use **str(a)** format to print it

3) print(calculation)

```
>>> print(5 * 3 - 1)

14
```

Note: Basic operators:

Operator	Description	Example
( )	Parentheses	print((1+2)*2) 6
+	Plus sign	print (1+4) 5
-	Subtraction sign	print(5-3) 2
*	Multiplication sign	print(9*4) 36
/	Division sign (gives exact value)	print(5/2) 2.5
//	Floor division (gives value without decimal part)	print(5//2) 2
%	Modulus sign (gives the remainder of the division)	print(5%2) 1
**	Exponent sign	print(5**2) 25

sqrt()	Square root sign
	(You can't use it without importing math module)
		from math import *
		print(sqrt(4))
		2

Note:

You can use multiply operator “ * “ between numbers and strings to print repeated text:

```
>>> print(“O” * 4)
OOOO
```

4) **Combination :** You can make combinations using all previous ways (1,2,3)

Combination operators:

Operator	Description	Example
+	Add strings together (without space)	print(“Hello”+”OjjOj”) HelloOjjOj
,	Add strings together (with space)	print(“Hello”,”OjjOj”) Hello OjjOj

Combination examples:

```
>>> a = 18 ;  b = “OjjOj”
>>> print (“My name is”, b , ”I am”, str(a) )
My name is OjjOj I am 18
```

Strings literals:

There are some characters that can’t be displayed without the use of ‘ \ ‘ before them:

Symbol	Description	Example
\”	Print “	print(“Hello \” OjjOj”) Hello “ OjjOj
\’	Print ‘	print(“Hello \’ OjjOj”) Hello ‘ OjjOj
\n	print new line character	print(“Hello \n OjjOj”) Hello OjjOj
\t	print tab character	print(“Hello \t OjjOj”) Hello OjjOj
\r	Print return character	print(“Hello \r OjjOj”) OjjOj
\$	Print $	print(“Hello \$ OjjOj”) Hello $ OjjOj
\\	Print \	print(“Hello \\ OjjOj”) Hello \ OjjOj

Note:

Types of objects you can use *print()* on :

```
>>> print(42)                          # <class 'int'>
42
>>> print(3.14)                        # <class 'float'>
3.14
>>> print(1 + 2j)                      # <class 'complex'>
(1+2j)
>>> print(True)                        # <class 'bool'>
True
>>> print([1, 2, 3])                   # <class 'list'>
[1, 2, 3]
>>> print((1, 2, 3))                   # <class 'tuple'>
(1, 2, 3)
>>> print({'red', 'green', 'blue'})    # <class 'set'>
{'red', 'green', 'blue'}
>>> print({'name': 'Ali', 'age': 42})  # <class 'dict'>
{'name': 'Ali', 'age': 42}
>>> print('hello')                     # <class 'str'>
hello
```

II – Intermediate uses:

Parameters: You can add (one or more) parameters to *print()* function to improve your printing (separating, ending...)

```
>>> print( statement , parameter = ... )
```

Parameter	Code	Description	Example
end	, end = “text”	Prints “text” at the last	print(“N”+”O”, end=”!!”) NO!!
sep	, sep = “text”	Separates the strings with “text” (It does not work if you using “ + ” between strings)	print(“Y”,”E”,”S”, sep=”\n”) Y E S

file	, file = file	Prints in an external file	new = open('file.txt', 'w')
			print('OjjOj', file = new)
			new.close()

flush	, flush = bool (True / False)	Flushes the internal buffer, like “stdio's fflush”. This may be a no-op on some file-like objects. (Default value False)	print(“Hello OjjOj”, flush = True)

III – Advanced uses:

Formatting:

Formatting is a great way to well arrange your print statement including variables.

1) { } format:

Form:

```
>>> print( “text {0} text {1} ...” .format(var1,var2...) )
```

In the curly brackets you put the variable index depending on variables you write inside: .format() (any variable type)

Exp:

```
>>> a = “OjjOj”

>>> b = 19

>>> print( “ Hello {0} you are {1} ” .format(a, b) )

Hello OjjOj you are 19
```

Note:

1) If you leave the curly brackets without any index number, the print function will display the variables in the same order as in: .format(**var1,var2...**)

Exp:

```
>>> a = “OjjOj”

>>> b = 19

>>> print( “ Hello {} you are {} ” .format(a, b) )

Hello OjjOj you are 19
```

2) You can fill in the curly brackets with indexes in any order you want.

Exp:

```
>>> a = “OjjOj”

>>> b = 19

>>> print( “Hello {1} you are {0} ” .format(a, b) )

Hello 19 you are OjjOj
```

3) You can fill: .format(**var1,var2...**) with values instead of variables.

Exp:

```
>>> print( “ Hello {} you are {} ” .format(“OjjOj” , 19) )

Hello OjjOj you are 19
```

4) You can use any variable, as many times as you want.

Exp:

```
>>> print( " Hello {0} you are {0} " .format("OjjOj") )
Hello OjjOj you are OjjOj
```

Format with parameters:

You can use a lot of parameters on format to let it do stuff like spacing, aligning, converting...

Form:

```
>>> print( "text {index: parameters} ..."
.format(var1,var2...) )
```

Exp:

```
>>> print( "{0: *^15}" .format("OjjOj") )
*****OjjOj*****
```

Parameters:

Form:

```
{index: FILL ALIGN SPACES CUT CONVERT}
```

Exp:

```
{0: *^14.6s}
```

△

You can't change parameters order

Note:

1) Never put spaces between parameters
2) You may not use all the parameters, leave the place of those not in use empty without space

```
{:15}
```

CONVERT:

Convert is used to convert a type of variable to another (exp: int to float)

Symbol:

s	*– string*
d	*– decimal integer*
f	*– float*
c	*– character*
b	*– binary*
o	*– octal*
x	*– hexadecimal with lowercase letters after 9*
X	*– hexadecimal with uppercase letters after 9*
e	*– exponent notation*

Exp:

```
>>> print( “I am {0:f}” .format(19) )
I am 19.000000
```

CUT:

You can print a part of the variable using this **CUT** parameter

Form:

`{.int}` **int:** number of characters you want to keep (count from left to right)

Exp:

```
>>> print( “{0:.3}” .format(“OjjOj”) )
Ojj
```

This parameter is usually used on float numbers:

int: number of digits you want to print (digits = before decimal point + after decimal point)

Exp:

```
>>> print( “{0:.3}” .format(2.81239) )
2.81
```

SPACES:

Space parameter takes **int** value, it makes an area with that amount of spaces, then prints the variable inside it. (It prints the variable aligned to the right by default)

Exp:

```
>>> print( "I am{0:10}" .format(19) )

I am        19
```

Note:

The variable length is considered within space:

(in the example above): between 'am' and '19' : 8 spaces

Space parameter value = 10 = 8 spaces + variable length

ALIGN:

Align the variable to the (right, center, left)

Symbol:

<	–	left-align text in the field
^	–	center text in the field
>	–	right-align text in the field

Note:

To align variable, you need some space, so using Align parameter without Spaces parameter is useless

Exp:

```
>>> print( “{0:<10}” .format(19) )

>>> print( “{0:^10}” .format(19) )

>>> print( “{0:>10}” .format(19) )

19

    19

        19
```

FILL:

Fill parameter is used to fill in the blank with a **character**: (a,b,c,d....1,2,3,4....-,+,*,/...)

Exp:

```
>>> print( “{0:*^15}” .format(“OjjOj”) )

*****OjjOj*****
```

Note:

1) You can only choose one character to fill in the blank
2) Using the Fill parameter needs some space to fill in, so using **Fill parameter** without some **Spaces parameter** is useless
3) You can't use the **Fill parameter** without aligning the variable to a specific direction: **Align parameter**

....

Exps:

1)

```
print("{:->10}{:->10}{:->10}"
.format(1,2,3))
```

Output: ---------1---------2---------3

2)

```
for i in range(10):
    print("{0:5}{1:5}{2:5}".format(i,
i**2,i**3))
```

Output:

```
    0    0    0
    1    1    1
    2    4    8
    3    9   27
    4   16   64
    5   25  125
    6   36  216
    7   49  343
    8   64  512
    9   81  729
```

3)

```
for i in range(10):
    print("{:_^9}".format(i))
```

Output:

```
____0____
____1____
____2____
____3____
____4____
____5____
____6____
____7____
____8____
____9____
```

……………………………………………………………

You can see more about the string formatting in the link below:

https://pyformat.info/

2) % formating:

Form:

```
>>> print( “text %d text %s ...”  %(var1, var2...) )
```

Using the % format : You can print variables by using % symbol and placing variable format (integer, float, string...) that you want to print the variable with, right after the % symbol.

The variables are passed at the end inside: %()

Exp:

```
>>> a = “OjjOj”

>>> b = 19

>>> print( “ Hello %s you are  %d ” %(a, b) )

Hello OjjOj you are 19
```

The variable format Symbols:

%d – integer (digit)
%f – float
%s – string
%x – hexadecimal
%o – octal

Note:

1) You should print the variables in the same order as passed in: %()

Exp:

```
>>> a = "OjjOj"

>>> b = 19

>>> print( " Hello %d you are  %s " %(a, b) )

TypeError
```

2) You can convert the type of the variable from one type to another using the **“Variable format symbols”**

Exp:

```
>>> a = “OjjOj”

>>> b = 19

>>> print( “ Hello %s you are  %f ” %(a, b) )

Hello OjjOj you are 19.000000
```

....

You can see more about the string formatting in the link below:

https://pyformat.info/

PyNOOB

SUBSCRIBE