Python workbook

Complete Python for beginners

Index

Python print() Function

Definition and Usage

The print() function prints the specified message to the screen, or other standard output device.

The message can be a string, or any other object, the object will be converted into a string before written to the screen.

Syntax

```
print(object(s), sep=separator, end=end, file=file, flush=flush)
```

Parameter Values

Parameter	Description
object(s)	Any object, and as many as you like. Will be converted to string before printed
sep='*separator*'	Optional. Specify how to separate the objects, if there is more than one. Default is ' '
end='*end*'	Optional. Specify what to print at the end. Default is '\n' (line feed)
file	Optional. An object with a write method. Default is sys.stdout
flush	Optional. A Boolean, specifying if the output is flushed (True) or buffered (False). Default is False

```
x = ("apple", "banana", "cherry")
print(x)
```

```
('apple', 'banana', 'cherry')
```

Python input() Function

Definition and Usage

The input() function allows user input.

Syntax

input(*prompt*)

Parameter Values

Parameter	Description
prompt	A String, representing a default message before the input.

Example

Ask for the user's name and print it:

```
print("Enter your name:")
x = input()
print("Hello, " + x)
```

```
Enter your name:
```

Python If ... Else

Python Conditions and If statements

Python supports the usual logical conditions from mathematics:

- Equals: a == b
- Not Equals: a != b
- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b

These conditions can be used in several ways, most commonly in "if statements" and loops.

An "if statement" is written by using the if keyword.

Example

If statement:

```
a = 33
b = 200

if b > a:
  print("b is greater than a")
```

```
b is greater than a
```

In this example we use two variables, a and b, which are used as part of the if statement to test whether b is greater than a. As a is 33, and b is 200, we know that 200 is greater than 33, and so we print to screen that "b is greater than a".

Indentation

Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly-brackets for this purpose.

Example

If statement, without indentation (will raise an error):

```
a = 33
b = 200

if b > a:
print("b is greater than a")
```

```
  File "demo_if_error.py", line 4
    print("b is greater than a")
        ^
IndentationError: expected an indented block
```

Elif

The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".

Example

In this example a is equal to b, so the first condition is not true, but the elif condition is true, so we print to screen that "a and b are equal".

```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

```
a and b are equal
```

Else

The else keyword catches anything which isn't caught by the preceding conditions.

In this example a is greater than b, so the first condition is not true, also the elif condition is not true, so we go to the else condition and print to screen that "a is greater than b".

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

```
a is greater than b
```

Short Hand If

If you have only one statement to execute, you can put it on the same line as the if statement.

Example

One line if statement:

```
a = 200
b = 33

if a > b: print("a is greater than b")
```

```
"a is greater than b"
```

Short Hand If ... Else

Example

One line if else statement:

```
a = 2
b = 330

print("A") if a > b else print("B")
```

```
B
```

And

The and keyword is a logical operator, and is used to combine conditional statements:

Example

Test if a is greater than b, AND if c is greater than a:

```
a = 200
b = 33
c = 500
if a > b and c > a:
  print("Both conditions are True")
```

```
Both conditions are True
```

Or

The or keyword is a logical operator, and is used to combine conditional statements:

Example

Test if a is greater than b, OR if a is greater than c

```
a = 200
b = 33
c = 500
if a > b or a > c:
  print("At least one of the conditions is True")
```

```
At least one of the conditions is True
```

Nested If

You can have if statements inside if statements, this is called *nested* if statements.

Example

```
x = 41

if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

```
Above ten,
and also above 20!
```

The pass Statement

if statements cannot be empty, but if you for some reason have an if statement with no content, put in the pass statement to avoid getting an error.

Example

```
a = 33
b = 200

if b > a:
  pass

# having an empty if statement like this, would raise an error without the pass statement
```

Python While Loops

Python Loops

Python has two primitive loop commands:

- while loops
- for loops

The while Loop

With the while loop we can execute a set of statements as long as a condition is true.

Example

```
i = 1
while i < 6:
  print(i)
  i += 1
```

```
1
2
3
4
5
```

Note: remember to increment i, or else the loop will continue forever.

The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, i, which we set to 1.

The break Statement

With the break statement we can stop the loop even if the while condition is true:

Example

Exit the loop when i is 3:

```
i = 1
while i < 6:
  print(i)
  if (i == 3):
    break
  i += 1
```

```
1
2
3
```

The continue Statement

With the continue statement we can stop the current iteration, and continue with the next:

Example

Continue to the next iteration if i is 3:

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)

# Note that number 3 is missing in the result
```

```
1
2
4
5
6
```

The else Statement

With the else statement we can run a block of code once when the condition no longer is true:

Example

Print a message once the condition is false:

```
i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less than 6")
```

```
1
2
3
4
5
i is no longer less than 6
```

Python For Loops

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

Example

Print each fruit in a fruit list:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

```
apple
banana
cherry
```

The for loop does not require an indexing variable to set beforehand.

The break Statement

With the break statement we can stop the loop before it has looped through all the items:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```

```
apple
banana
```

The continue Statement

With the continue statement we can stop the current iteration of the loop, and continue with

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    continue
  print(x)
```

```
apple
cherry
```

the next:

The range() Function

To loop through a set of code a specified number of times, we can use the range() function,

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

```
for x in range(6):
  print(x)
```

```
0
1
2
3
4
5
```

ote that range(6) is not the values of 0 to 6, but the values 0 to 5.

The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(2, 6), which means values from 2 to 6 (but not including 6):

Else in For Loop

The else keyword in a for loop specifies a block of code to be executed when the loop is finished:

```
for x in range(6):
  print(x)
else:
  print("Finally finished!")
```

```
0
1
2
3
4
5
Finally finished!
```

Nested Loops

A nested loop is a loop inside a loop.

The "inner loop" will be executed one time for each iteration of the "outer loop":

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)
```

```
red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry
```

The pass Statement

for loops cannot be empty, but if you for some reason have a for loop with no content, put in the pass statement to avoid getting an error.

```
for x in [0, 1, 2]:
  pass

# having an empty for loop like this, would raise an error without the pass statement
```

END OF Chapter 5