
Python Programming
Flow Control

Flow Control

- if conditions in Python.
- Loops in Python.
- Generator functions.
- List comprehensions.

Conditional Statements

- if statement

```
if some_conditions:
    do_this_1()
    do_this_2()
do_this_after()
```

- elif and else

```
if some_conditions:
    do_this_1
else:
    do_this_2()

if some_conditions:
    do_this_1
elif other_conditions:
    do_this_2
else:
    do_this_3() do_this_after()
```

- False =
  - None
  - 0 (or 0.0)
  - empty list, tuple, etc
- use bool() to verify, e.g.:
  - bool(None)

Comparison operators

Operator	Description
==	equals
!=	doesn't equal
>	is greater than
<	is less than
>=	is greater than or equal to
<=	is less than or equal to
is	is the same object
is not	is not the same object

not in

not in

- True if value is not found in sequence.

```
'a' not in ['a', 'b', 'c'] # False
'd' not in 'abc' # True
```

- Demo 35: flow-control/Demos/if.py

Compound conditions

- and or and not
- Demo 36: flow-control/Demos/if2.py

is and is not

- different from == and !=

```
>>> a = [1, 2, 3, 4]
>>> b = [1, 2, 3, 4]
>>> c = a
>>> a == b
True
>>> a is b
False
>>> a == c
True
>>> a is c
True
>>> id(a), id(b), id(c)
(1847278372096, 1847279744512, 1847278372096)
>>> a.append(5)
>>> c
[1, 2, 3, 4, 5]
```

- a and c are the same object

all() and any()

- used to loop through an iterable to check the boolean value of its items
- The all() function returns True if all the elements of the iterable evaluate to True.
- The any() function returns True if any of the elements of the iterable evaluate to True.

```
>>> print(all([1,2,3]))
True
>>> print(all([1,2,3,0]))
False
>>>
```

Ternary operator

- shorthand for if-else

```
[on_true] if [expression] else [on_false]
```

- for example

```
pant_color = 'white' if season == 'summer' else 'black'
```

in between

- check if in between two values

```
if low_value < your_value < high_value:
```

- for example

```
age = 15

if 12 < age < 20:

    print("You are a teenager.")

if 13 <= age <= 19:

    print("You are a teenager.")
```

Loops

- while loops

```
num=0
while num < 6:
    print(num)
    num += 1
```

- Demo 37: flow-control/Demos/while_input.py
- Demo 38: flow-control/Demos/guess_the_number.p

For loops

```
for item in sequence_name:
    do_something(item)
```

- looping through a range

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

for num in range(0, 6):
    print(num)
```

- skipping steps

```
for num in range(1, 11, 2): # 3rd argument is the step
    print(num)
```

For loops

- stepping backwards

```
for num in range(10, 0, -1):
    print(num)
```

- loop through list / tuple

```
nums = [0, 1, 2, 3, 4, 5]
for num in nums:
    print(num)

nums = (0, 1, 2, 3, 4, 5)
for num in nums:
    print(num)
```

For loops

- loop through dictionaries

```
grades = {'English':97, 'Math':93, 'Art':74, 'Music':86}

for course in grades:
    print(course)

for course in grades.keys():
    print(course)
```

- loop through dict values

```
for grade in grades.values():
    print(grade)
```

For loops

- loop through dict items:

```
for course, grade in grades.items():
    print(f'{course}: {grade}')
```

- dict_items is a pseudo tuple

```
dict_items([('English', 97), ('Math', 93), ('Art', 74), ('Music', 86)]
```

- another way is

```
for item in grades.items():
    course = item[0]
    grade = item[1]
    print(f'{course}: {grade}')
```

Exercise 22

- All True and Any True
  - 10-15 min

break and continue

- break out of loop

```
for num in range(11, 20):
    print(num)
    if num % 5 == 0:
        break
```

- jump to next iteration

```
for num in range(1, 12):
    if num % 5 == 0:
        continue
    print(num)
```

- Demo 39: flow-control/Demos/guess_the_number_continue.py
- Demo 40: flow-control/data/states.txt
- Demo 41: flow-control/Demos/get_state.py

Exercise 23

- Word Guessing Game
  - 45-120 min

else and loops

- in Python, for and while loops have an optional else clause, which is executed after the loop has successfully completed iterating
- Demo 42: flow-control/Demos/loop_else.py
- when would you use this?
  - e.g. to check user-entered text for black-listed words
- without for–else

```
sentence = input('Input a sentence: ')
found_bad_word = False
for word in sentence.split():
    if word in BLACK_LIST:
        found_bad_word = True
    break
if found_bad_word:
    print('You used a naughty word.')
else:
    print('Sentence passes cleanliness test.')
```

- with for-else: Demo 43: flow-control/Demos/loop_else_use_case.py

Exercise 24

- for...else
  - 10-20 min

The enumerate() function

- Demo 44: flow-control/Demos/without_enum.py
- Demo 45: flow-control/Demos/with_enum.py
- enumerate() takes two arguments:
  - 1. The iterable to enumerate.
  - 2. The number at which to start counting (defaults to 0).
- it returns an iterable of two-element tuples of the format (count, value).
- Demo 46: flow-control/Demos/state_list.py

Generators

- A generator is a function that returns an object (iterator) which we can iterate over (one value at a time)

Create Generators in Python

- It is fairly simple to create a generator in Python. It is as easy as defining a normal function, but with a yield statement instead of a return statement.
- If a function contains at least one yield statement (it may contain other yield or return statements), it becomes a generator function. Both yield and return will return some value from a function.
- The difference is that while a return statement terminates a function entirely, yield statement pauses the function saving all its states and later continues from there on successive calls.

Generators

- Demo 47: flow-control/Demos/list_loop.py
- Demo 48: flow-control/Demos/generator_loop.py
- if you want to iterate through multiple times, you can simply call the generator function again
  - Demo 49: flow-control/Demos/generator_loop2.py
- Use generators whenever you want to be able to get the next item in a sequence without creating the whole sequence ahead of time.
- Demo 50: flow-control/Demos/odd_numbers.py

Fibonacci sequence

- Demo 51: flow-control/Demos/fibonacci.py
- The Fibonacci sequence is a sequence in mathematics in which each number is the sum of the two preceding numbers.

A Sequence of Unknown Length

- Demo 52: flow-control/Demos/bingo.py
- This generator function keeps spitting out Bingo balls until it runs out of balls.
- Try creating a bingo card with a random function and play Bingo using this program

B	I	N	G	O
9	24	31	58	69
11	19	34	57	62
15	25	●	60	72
14	17	38	47	68
6	23	33	51	74

next() function

- To be an iterator, an object must have a special __next__() method (that's two underscores before and two underscores after "next"), which returns the next item of the iterator.
- With a generator, methods like __iter__() and __next__() are implemented automatically. So we can iterate through the items using next().
- Python's built-in next(iter) function calls the __next__() method of iter. The following example shows how to use the next() function with a generator to play Rock, Paper, Scissors against your computer.
- Demo 53: flow-control/Demos/roshambo.py

List comprehensions

- List comprehensions are used to create new lists from existing sequences by taking a subset of that sequence and/or modifying its members. Syntax:

```
new_list = [modify(item) for item in items]
```

- creating a new list using a loop:

```
squares = []
for i in range(10):
    squares.append(i*i)
print(squares) # [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

- using list comprehension:

```
squares = [i*i for i in range(10)
```

List comprehension

- Demo 54: flow-control/Demos/list_from_list
  - The list comprehensions creates a new list from names by passing each of its elements to the initials() function.
- Demo 55: iterables/Demos/add_nums.py
  - We can improve the output with list comprehension:
  - Demo 56: flow-control/Demos/add_nums.p

Creating sublists

- List comprehensions have an optional if clause that can be used to create a sublist from a list:

```
new_list = [item for item in items if some_condition]
```

- get 3-letter words from a list:

```
three_letter_words = []
for word in words:
    if len(word) == 3:
        three_letter_words.append(word)
```

- with list comprehension:
  - Demo 57: flow-control/Demos/sublist_from_list.py
  - combines an if condition within a for loop into a single line.

List comprehension

- Demo 58: flow-control/Demos/states_with_spaces.py
  - This reads in the content of the states.txt file and then splits it into a list of lines. It then loops through that list to create a new list of just the states that have spaces in their names.
- Demo 59: flow-control/Demos/states_with_spaces2.py
  - replace the for loop with a couple of list comprehensions
- could also be done in a single step
  - Demo 60: flow-control/Demos/states_with_spaces3.py
  - While that’s a bit harder to read, the code is more efficient, because it doesn’t have to loop through lines and then loop again through states. It just does all the work the first time through the loop.