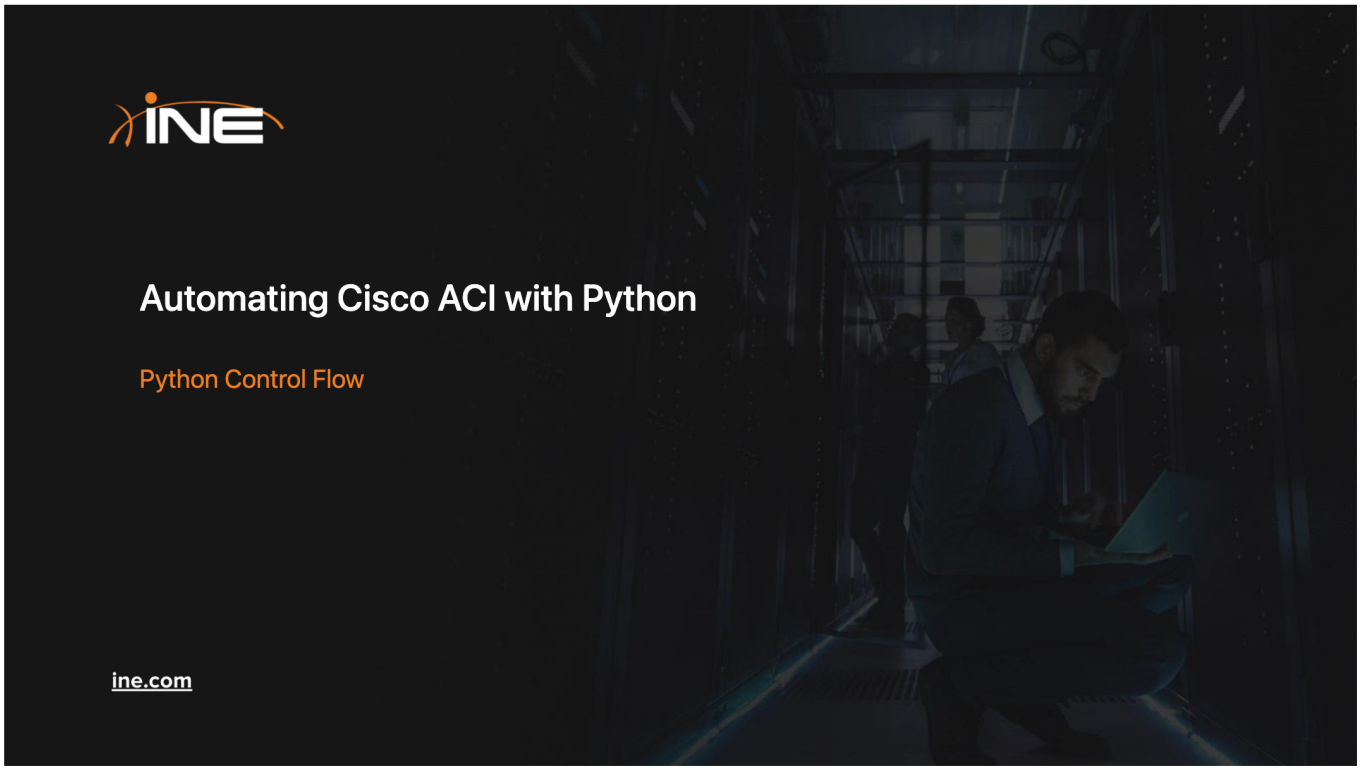
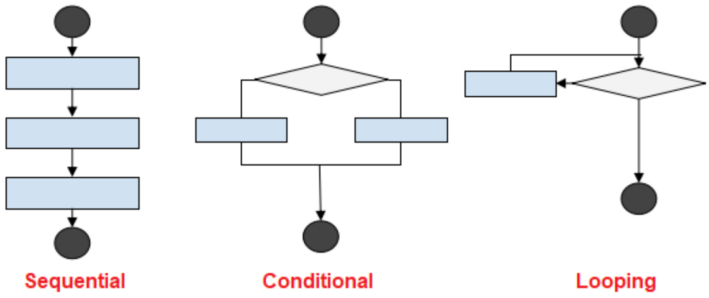


Another important topic when learning Python is **Control flow**. Python supports the most common control flow blocks. Keep in mind they're defined with indentation.

Control flow is the logical execution order statements or expressions of our algorithm will follow. Python has different ways in which the statements could be executed.

There are three main ways of execution:



> You can visualize code execution using Python Tutor.

Sequential flow control

Python starts reading at the top and keeps going down for each one of these statements:

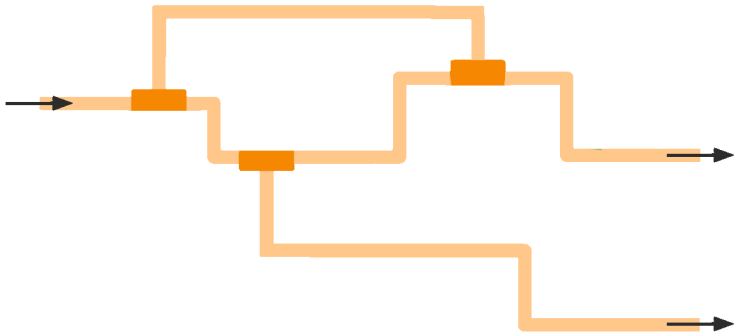
In [157…
```
print('1')
print('2')
print('3')
print('4')
print('5')
```

```
1
2
3
4
5
```

Python give us the posibility to change the sequential flow control from above by using different **Control flow structures**:

- Conditionals (`if` , `else` , `elif`)
- Loops (`while` , `for`)
- `return`
- Exceptions (`raise`)

So when you're writting your algorithm, you can combine these flow structures to create complex execution flows.

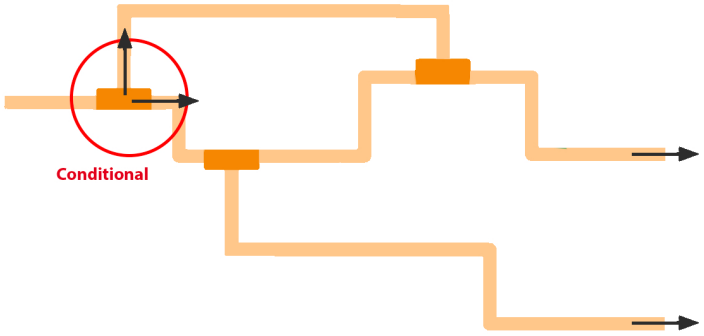


Let's see some examples.

Conditionals (*if*, *else*, *elif*)

Our algorithm now will evaluate our condition and execute statement(s) only if the condition is `True` .

If the condition is `False` , the statement(s) is not executed.



Let's create our first conditional flow by defining one simple age condition:

In [158…
```
age = 35

if age >= 18:
    # +18 years old
    print("You can drive!")

print("...end program")
```

```
You can drive!
...end program
```

In [159…
```
age = 14

if age >= 18:
    # +18 years old
    print("You can drive!")

print("...end program")
```

```
...end program
```

We can improve our algorithm by adding an `else` statement after our `if` .

The `if..else` statement evaluates the condition and will execute the body of `if` only when the condition is `True` as before.

But now -thanks to the `else` statement- if the condition is `False`, the body of `else` will be executed.

Let's see an example:

In [160…
```
age = 14

if age > 18:
    # +18 years old
    print("You can drive!")
else:
    # 0-17 years old
    print("You can't drive...")

print("...end program")
```

```
You can't drive...
...end program
```

Python interprets non-zero values as `True`. `None` and `0` are interpreted as `False`.

In [161…
```
credits = 10

if credits > 0:
    print("You can play")
else:
    print("Buy more credits to play")

print("...end program")
```

```
You can play
...end program
```

Also, you can add nested conditions as needed:

In [162…
```
credits = 10

if credits > 0:
    # 1, 2, 3,... N credits
    print("You can play")

    if credits == 1:
        # 1 credit
        print("This is your last play!")
else:
    print("Buy more credits to play")

print("...end program")
```

```
You can play
...end program
```

In [163…
```
credits = 1

if credits > 0:
    # 1, 2, 3,... N credits
    print("You can play")

    if credits == 1:
        # 1 credit
        print("This is your last play!")
else:
    print("Buy more credits to play")

print("...end program")
```

```
You can play
This is your last play!
...end program
```

What happens if we have `credits = -5`?

In [164…
```
credits = -5

if credits > 0:
    # 1, 2, 3,... N credits
    print("You can play")

    if credits == 1:
        # 1 credit
        print("This is your last play!")
else:
    # 0, -1, -2.. credits
    print("Buy more credits to play")

print("...end program")
```

```
Buy more credits to play
...end program
```

In [165…
```
credits = -5

if credits > 0:
    # 1, 2, 3,... N credits
    print("You can play")

    if credits == 1:
        # 1 credit
        print("This is your last play!")
else:
    if credits == 0:
```

```
        # 0 credits
        print("Buy more credits to play")
    else:
        # -1, -2.. credits
        print("Error on your credits")

print("...end program")
```

```
Error on your credits
...end program
```

To avoid infinite nested `if` statements and improve our algorithm we can use the `elif` statement:

In [166…

```
credits = -5

if credits > 0:
    # 1, 2, 3,... N credits
    print("You can play")

    if credits == 1:
        # 1 credit
        print("This is your last play!")
elif credits == 0:
    # 0 credits
    print("Buy more credits to play")
else:
    # -1, -2.. credits
    print("Error on your credits")

print("...end program")
```

```
Error on your credits
...end program
```

This `elif` will be more appropriate if you have multiple exclusive conditions:

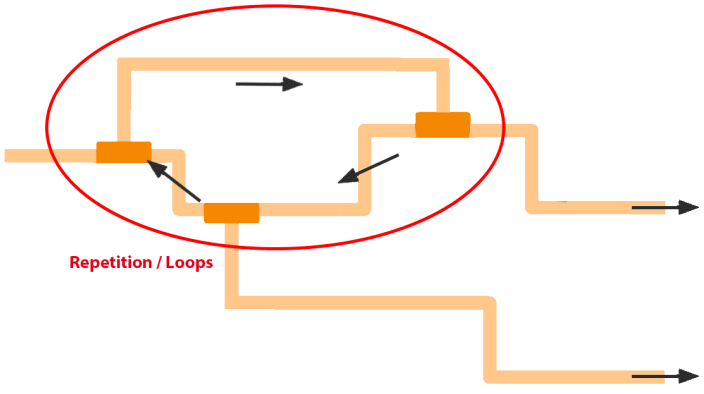
In [167…

```
credits = 12

if credits > 15:
    # +15 credits
    print("You have more than 15 credits left")
elif credits > 10:
    # 11-15 credits
    print("You have more than 10 credits left")
elif credits > 5:
    # 6-10 credits
    print("You have more than 5 credits left")
elif credits > 0:
    # 1-5 credits
    print("Low credits!")
elif credits == 0:
    print("Buy more credits to play")
else:
    # -1, -2.. credits
    print("Error on your credits")

print("...end program")
```

```
You have more than 10 credits left
...end program
```

Loops (*for*, *while*)

Loop control statements change execution from its normal sequence.



Python programming language provides following types of loops to handle looping requirements. We will focus on:

- `for` loop
- `while` loop

for loop

The `for` loop in Python is used to iterate over a collection/sequence (list, tuple, string) or other iterable objects.

Our algorithm will use a variable to take the value of the item inside the sequence on each iteration.

The loop sequence will continue until we reach the last item in the sequence.

Let's create our first loop:

In [168…
```
print('1')
print('2')
print('3')
print('4')
print('5')

print("...end program")
```

```
1
2
3
4
5
...end program
```

We need a structure which executes the following:

```
repeat for each element:
    statements / do something
```

In [169…
```
for i in [1,2,3,4,5]:
    print(i)

print("...end program")
```

```
1
2
3
4
5
...end program
```

In [170…
```
for i in range(1,6):
    print(i)
```

```
1
2
3
4
5
```

In [61]:
```
names = ['Monica', 'Ross', 'Chandler', 'Joey', 'Rachel']
```

In [62]:
```
for name in names:
    print(name)
```

```
Monica
Ross
Chandler
Joey
Rachel
```

In [171…
```
animals = ['dog', 'cat', 'duck']

for animal in animals:
    print(animal)

print("...end program")
```

```
dog
cat
duck
...end program
```

This `for` loop also works on strings:

In [172…
```
word = 'Python'

for letter in word:
    print(letter)

print("...end program")
```

```
P
y
t
h
o
n
...end program
```

while loop

The `while` structure will repeat a statement or group of statements while a given condition is `True`.

It tests the condition before executing the loop body.

```
repeat while condition is True:
    statements / do something
    (increment counter / take closer to break out)
```

> While loops are seldom used in Python. For loops are the preferred choice 99% of the time. Still, they're available and are useful for some situations

Let's see an example:

In [1]:
```
count = 0
```

In [2]:
```
while count < 3:
    print("Counting...")
    count += 1
```

```
Counting...
Counting...
Counting...
```

In [3]:
```
counter = 0

# while number of repetitions is less than 4:
while counter < 4:
    print('Repetition: ', counter)
    # counter = counter + 1
    counter += 1

print("...end program")
```

```
Repetition:  0
Repetition:  1
Repetition:  2
Repetition:  3
...end program
```

> It's super important to increment count on each repetition to avoid `counter` being `0` forever!

In [174…
```
counter = 1

# while number of repetitions is less or equal than 4:
while counter <= 4:
    print('Number: ', counter)
    counter += 1

print("...end program")
```

```
Number:  1
Number:  2
Number:  3
Number:  4
...end program
```

Exceptions

Exceptions are raised at runtime when an abnormal situation is produced in your program. Exceptions can also be constructed and raised by your code. Example of an exception:

In [4]:
```
age = "30"
```

In [5]:
```
if age > 21:
    print("Allowed entrance")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-5-31f2672e1932> in <module>
----> 1 if age > 21:
      2     print("Allowed entrance")

TypeError: '>' not supported between instances of 'str' and 'int'
```

Exceptions can be handled at runtime with a `try/except` block:

In [6]:
```
try:
    if age > 21:
        print("Allowed entrance")
except:
    print("Something went wrong")
```

```
Something went wrong
```

The `except` portion can receive also be parametrized with the expected exception:

In [7]:
```
try:
    if age > 21:
        print("Allowed entrance")
except TypeError:
```

```
except TypeError:
    print("Age is probably of a wrong type")
```

```
Age is probably of a wrong type
```

Another possibility is raising a value error if we detect something is wrong with the type:

In [16]:
```
if type(age) != int:
    raise TypeError("Age should be an integer")
elif age > 21:
    print("Allowed entrance")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-16-02fca5149d63> in <module>
      1 if type(age) != int:
----> 2     raise TypeError("Age should be an integer")
      3 elif age > 21:
      4     print("Allowed entrance")

TypeError: Age should be an integer
```

In [17]:
```
try:
    a = 4
    b = 0
    c = a/b
    d = list1[8]
    e = c**d
except TypeError:
    print("We got a TypeError", err)
    raise
except ValueError as err:
    print("We got a ValueError", err)
    raise
except IndexError as err:
    print("We got an IndexError", err)
    raise
except ZeroDivisionError as err:
    print("We divided by zero!")
    raise
except:
    print("I really don't know what went wrong")
    raise
```

```
We divided by zero!
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-17-bf09fe19b616> in <module>
      2     a = 4
      3     b = 0
----> 4     c = a/b
      5     d = list1[8]
      6     e = c**d

ZeroDivisionError: division by zero
```

Exercises

Exercise 1

Change the code between brackets (`[]`) to make the variable `message` equals to `"Hello World"` .

In [ ]:
```
message = None

# Remove square brackets and write condition to make if statement code run
if [...]:
    message = "Hello World"

print(message)
```

In [ ]:
```
# Solution

message = None

if True:
    message = "Hello World"

print(message)
```

Exercise 2

Complete the code in the editor to make `message_1` equal to `"Hello World"` and `message_2` equal to `"This is INE"` .

In [ ]:
```
x = 10
```

```
if x == ...:
    message_1 = "Hello World"

if x != ...:
    message_2 = "This is INE"

#print(message_1)
#print(message_2)
```

In [ ]:
```
# Solution

x = 10

if x == 10:
    message_1 = "Hello World"

if x != 11:
    message_2 = "This is INE"

print(message_1)
print(message_2)
```

Exercise 3

Complete the code on the editor to answer to the questions if 583 is divisible by 11 and 911 is divisible by 11.

> Hint: Check the "modulo" operator

In [ ]:
```
divisible_by_11 = 11

is_583_divisible_by_11 = None
is_911_divisible_by_11 = None

number_583 = 583

if ...:
    is_583_divisible_by_11 = True
else:
    is_583_divisible_by_11 = False


is_911_divisible_by_11 = None
number_911 = 911

# Remove square brackets and complete if statement condition
if ...:
    is_911_divisible_by_11 = True
else:
    is_911_divisible_by_11 = False
```

In [1]:
```
# Solution

divisible_by_11 = 11

is_583_divisible_by_11 = None
is_911_divisible_by_11 = None

number_583 = 583

if number_583 % divisible_by_11 == 0:
    is_583_divisible_by_11 = True
else:
    is_583_divisible_by_11 = False


is_911_divisible_by_11 = None
number_911 = 911

if number_911 % divisible_by_11 == 0:
    is_911_divisible_by_11 = True
else:
    is_911_divisible_by_11 = False
```