Lecture 04:

Control Structures

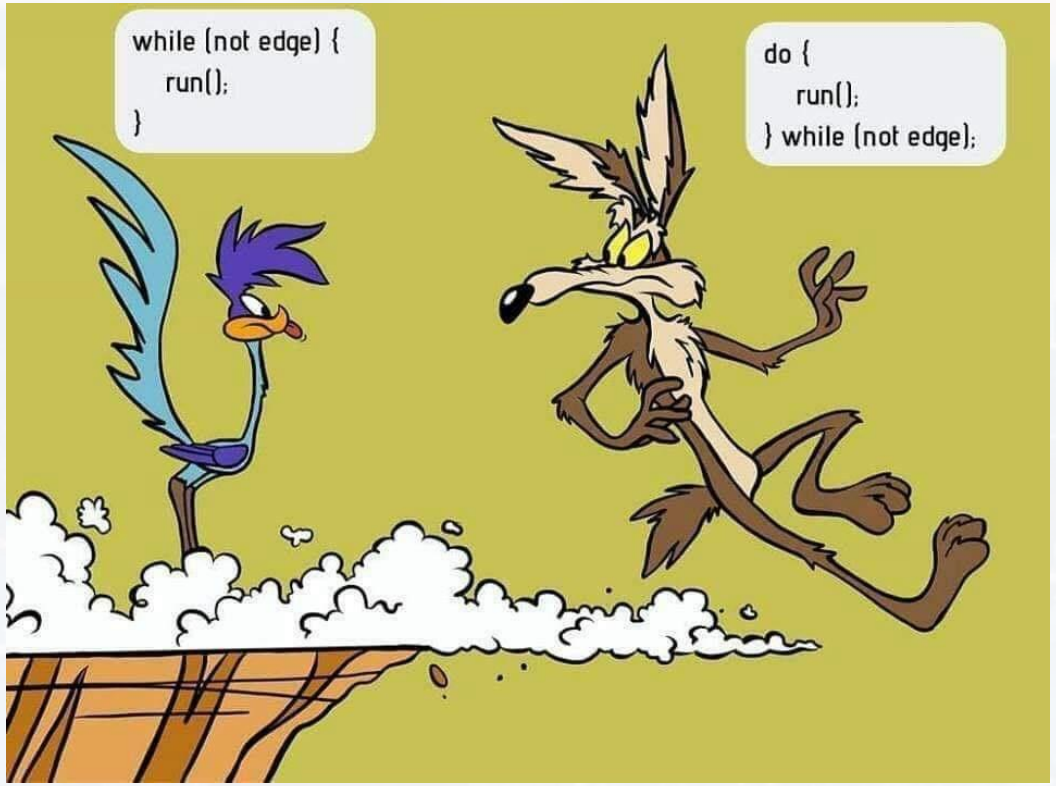
Markus Hohle

University California, Berkeley

Python for Molecular Sciences

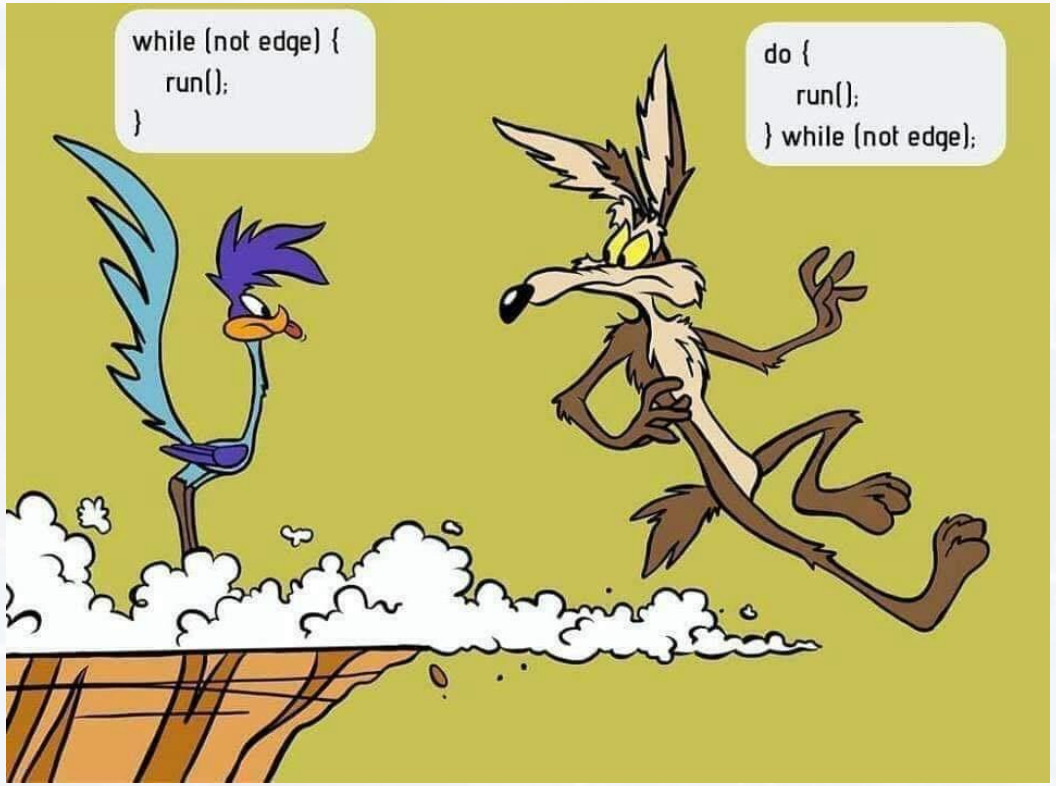
MSSE 272, 3 Units

Berkeley MSSE
MOLECULAR SCIENCE & SOFTWARE ENGINEERING



Outline

- General Idea and Structure
- `for` Loops and Comprehension
- `if`, `else` and `elif`
- `while`
- `break`, `continue` and `pass`



Outline

- General Idea and Structure

- `for` Loops and Comprehension
- `if`, `else` and `elif`
- `while`
- `break`, `continue` and `pass`

often (not always): iterating over an object

	numeric: `int`, `float`, `complex`	`5`, `5.55`, `(5+5j)`
iteratable	strings: `str`	`'this is a string'`, `"this is a string"`
	sequence: `list`, `tuple`, `range`	`my_tuple = (3, 'a', [2,3,4,5])` `range(10)`
mutable		`my_list = [1, 2, 'a']`
	mapping: `dict`	`my_dict = {1: 'a', 2: 'b'}`
	mapping: `set`	`my_set = {1, 2, 'a'}`
	boolean:	`True` `False`
	none type:	`None`
	callable: functions, methods, classes	`def`, `class`, `map`, `lambda`
	modules:	`from my_module import my_method as my_alias`

often (not always): iterating over an object

iterable

strings: `str`	`'this is a string', "this is a string"`
sequence: `list`, `tuple`, `range`	`my_tuple = (3, 'a', [2,3,4,5])` `range(10)`
mutable	`my_list = [1, 2, 'a']`
mapping: `dict`	`my_dict = {1: 'a', 2: 'b'}`
mapping: `set`	`my_set = {1, 2, 'a'}`

often (not always): iterating over an object

iteratable

strings: `str`

```
'this is a string', "this is a string"
```

sequence: `list`, `tuple`, `range`

```
my_tuple = (3, 'a', [2,3,4,5])
range(10)
my_list  = [1, 2, 'a']
```

mapping: `dict`

```
my_dict  = {1: 'a', 2: 'b'}
```

mapping: `set`

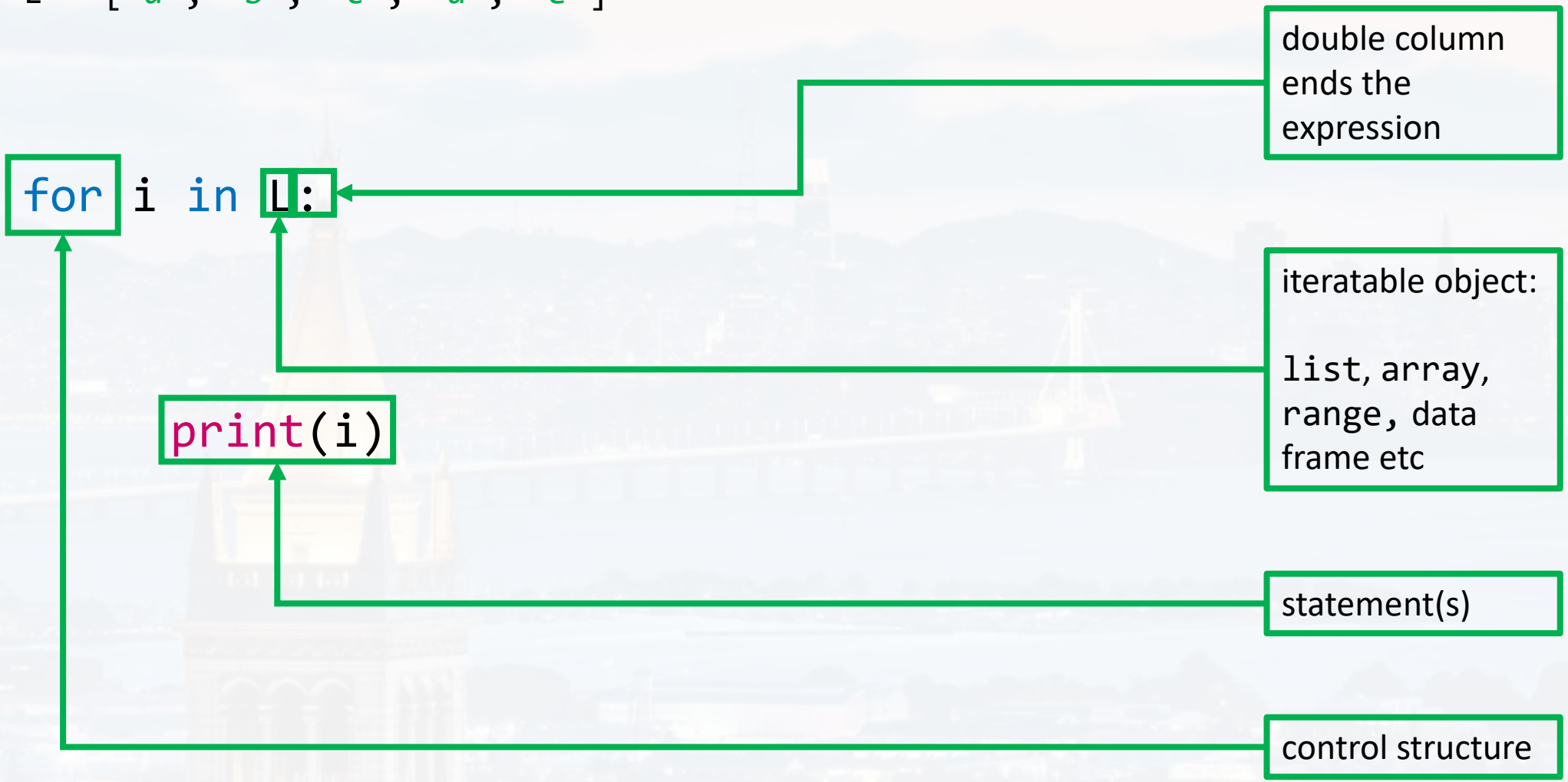
```
my_set   = {1, 2, 'a'}
```

when to use:
- repetitive operations
- distinguish between different cases/ parts of the DA pipeline

but:
- avoid (especially `for` and `while`) loops as far as possible
- loops are slow → vectorization (see later)

often (not always): iterating over an object

```
L = ['a', 'b', 'c', 'd', 'e']
```



often (not always): iterating over an object

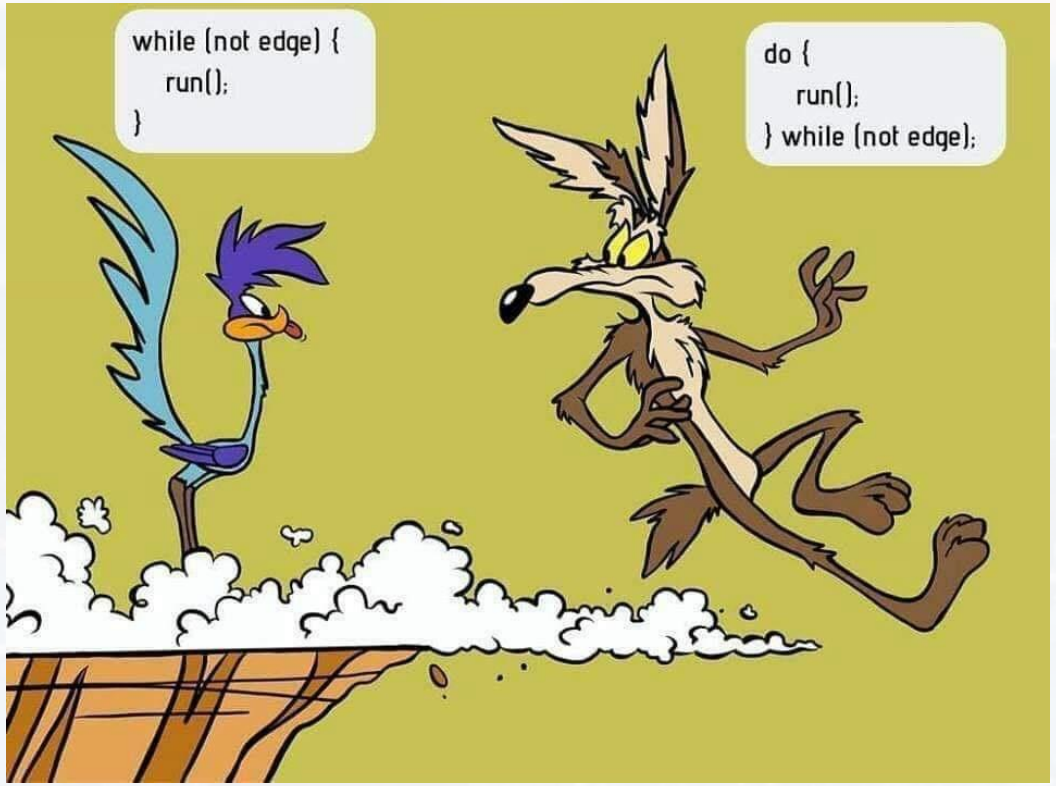
```
L = ['a', 'b', 'c', 'd', 'e']
```

```
for i in L:
    print(i)
```

free index variable

four blank spaces or tab

```
In [2]: L = ['a', 'b', 'c', 'd', 'e']

In [3]: for i in L:
   ...:
   ...: print(i)
   ...:
a
b
c
d
e
```



Outline

- General Idea and Structure
- **`for` Loops and Comprehension**
- `if`, `else` and `elif`
- `while`
- `break`, `continue` and `pass`

```
L = ['a', 'b', 'c', 'd', 'e']

for i in L:

    print(i)
```

iteratable object:

list, array, range, data frame etc

```
for i in 5:

    print(i)
```

vs

```
for i in range(5):

    print(i)
```

```
Traceback (most recent call last):

  Cell In[1], line 1
    for i in 5:

TypeError: 'int' object is not iterable
```

```
In [3]: for i in range(5):
   ...:
   ...:     print(i)
   ...:
0
1
2
3
4
```

```
L = ['a', 'b', 'c', 'd', 'e']
```

iterating over **content and index** of an object

```
for i, j in enumerate(L):
        print(str(i) + str(j))
```

```
0a
1b
2c
3d
4e
```

```
L = ['a', 'b', 'c', 'd', 'e']
```

iterating over **two** objects **simultaneously**

```
R = range(0,10,2)

for i, j in zip(R, L):

        print(str(i) + str(j))
```

```
0a
2b
4c
6d
8e
```

note: Even works, if objects have different lengths.
Just stops with the shortest object

```
L = ['a', 'b', 'c', 'd', 'e']
```

```
R = range(0,10,2)
```

iterating over **two** objects **simultaneously** and over **indices and content**

```
for i, (j, k) in enumerate(zip(R, L)):
        print(str(i) + str(j) + k)
```

```
00a
12b
24c
36d
48e
```

The more pythonic way is using ***comprehension***

```
from math import *

N = 10
Factorial = [None]*N

for n in range(N):
        Factorial[n] = factorial(n)
```

Index	Type	Size	
0	int	1	1
1	int	1	1
2	int	1	2
3	int	1	6
4	int	1	24
5	int	1	120
6	int	1	720
7	int	1	5040
8	int	1	40320
9	int	1	362880

comprehension

```
Factorial = [factorial(n) for n in range(N) ]
```

Index	Type	Size	
0	int	1	1
1	int	1	1
2	int	1	2
3	int	1	6
4	int	1	24
5	int	1	120
6	int	1	720
7	int	1	5040
8	int	1	40320
9	int	1	362880

note: very common in the Python community

shorter, often faster

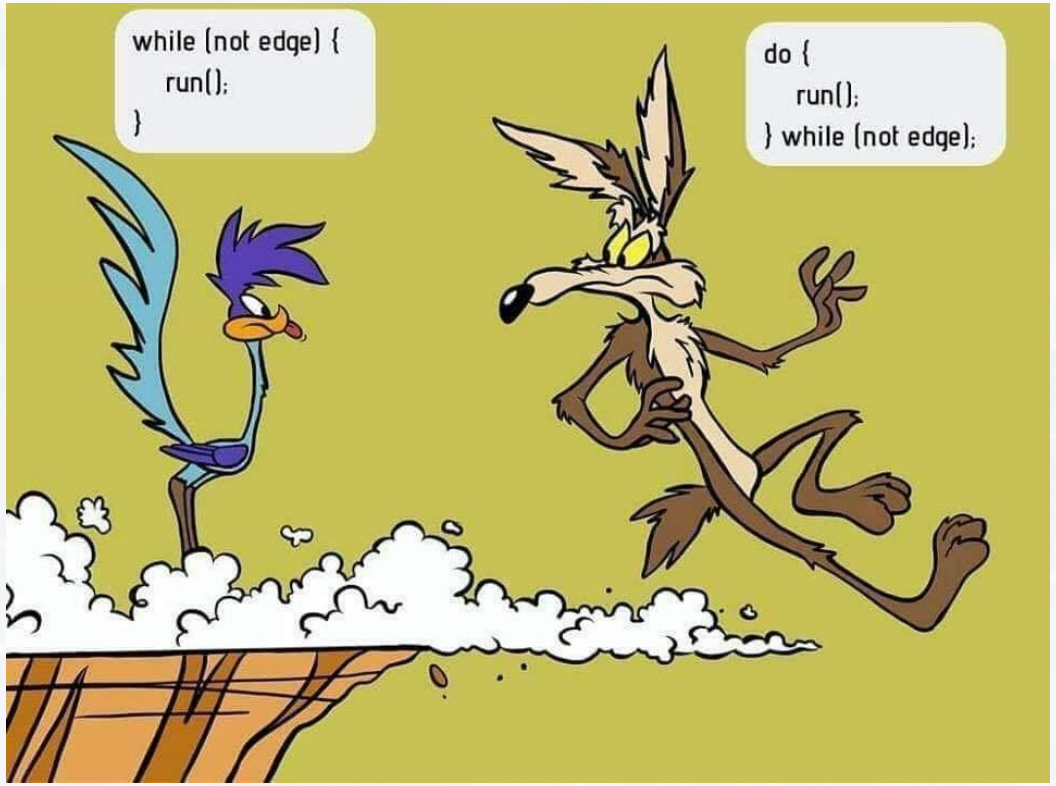
only if loops are not too complex

The more pythonic way is using *comprehension*

```
NT   = 'ACGT'
Code = [[1,0,0,0],
        [0,1,0,0],
        [0,0,1,0],
        [0,0,0,1]]
```

```
Dict = {}

for code, nt in zip(Code, NT):
        Dict[nt] = code

Dict['A']
```

```
Dict = {nt: code  for code, nt in zip(Code, NT) }

Dict['A']
```



Outline

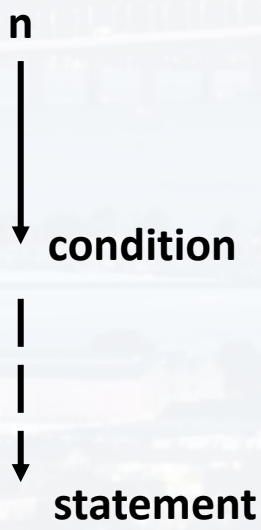
- General Idea and Structure
- `for` Loops and Comprehension
- **`if`, `else` and `elif`**
- `while`
- `break`, `continue` and `pass`

```
for n in range(10):
    if n%2 == 0:
        print('even number: ' + str(n))
```

applies if a condition is true

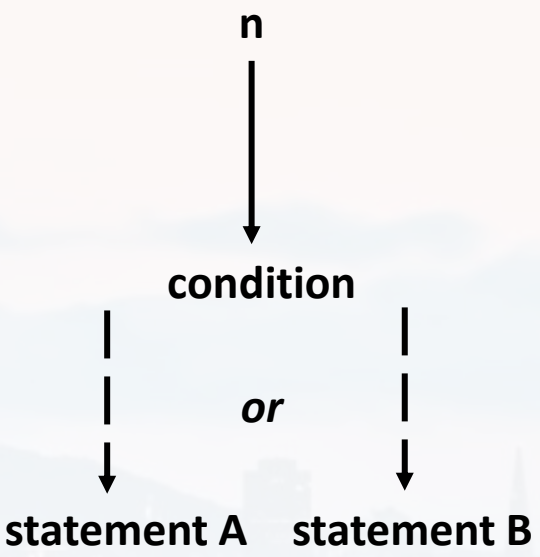
a logical condition (therefore ==)

```
even number: 0
even number: 2
even number: 4
even number: 6
even number: 8
```



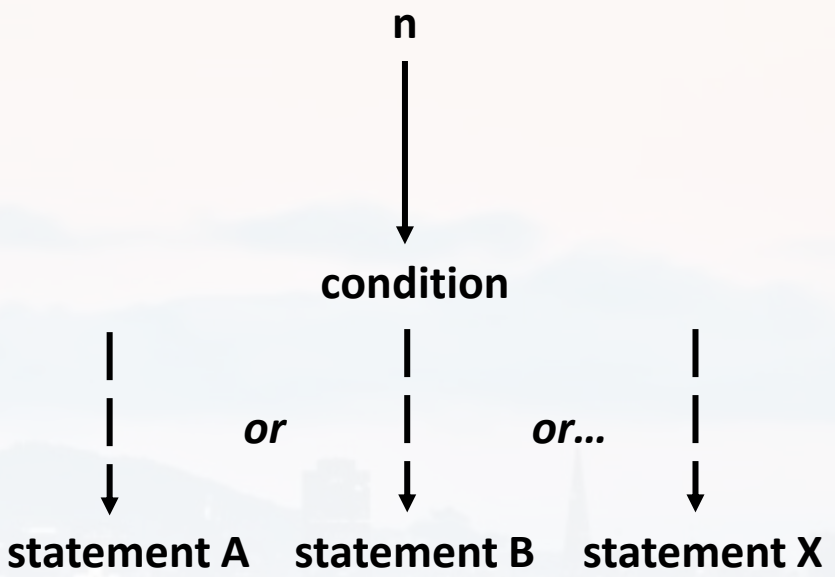
if, else and elif

```
for n in range(10):

    if n%2 == 0:
        print('even number: ' + str(n))

    else:
        print('odd number: ' + str(n))
```




```
even number: 0
odd number: 1
even number: 2
odd number: 3
even number: 4
odd number: 5
even number: 6
odd number: 7
even number: 8
odd number: 9
```

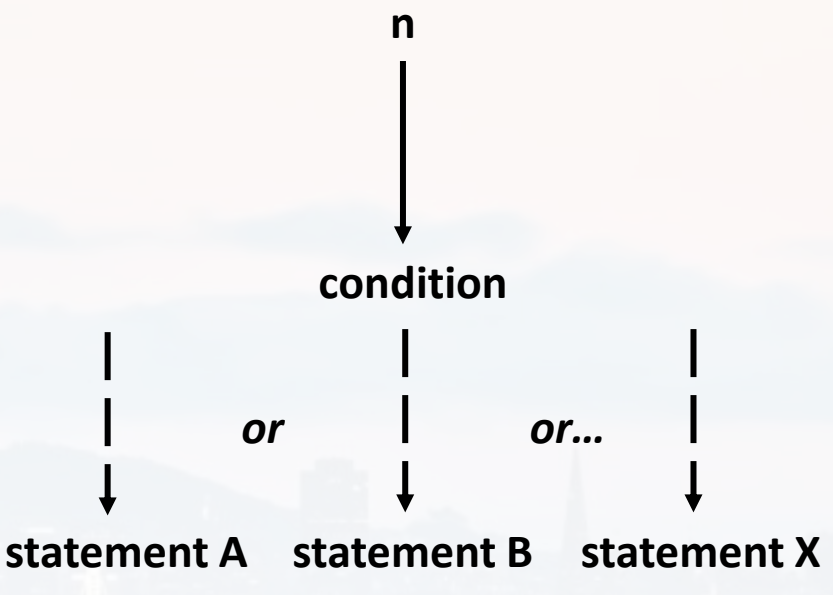
```
for n in range(1,10):

    if n%2 == 0:
        print('even number: ' + str(n))

    if n%3 == 0:
        print('multiple of 3: ' + str(n))

    if n%5 == 0:
        print('multiple of 5: ' + str(n))
```




```
even number: 2
multiple of 3: 3
even number: 4
multiple of 5: 5
even number: 6
multiple of 3: 6
even number: 8
multiple of 3: 9
```

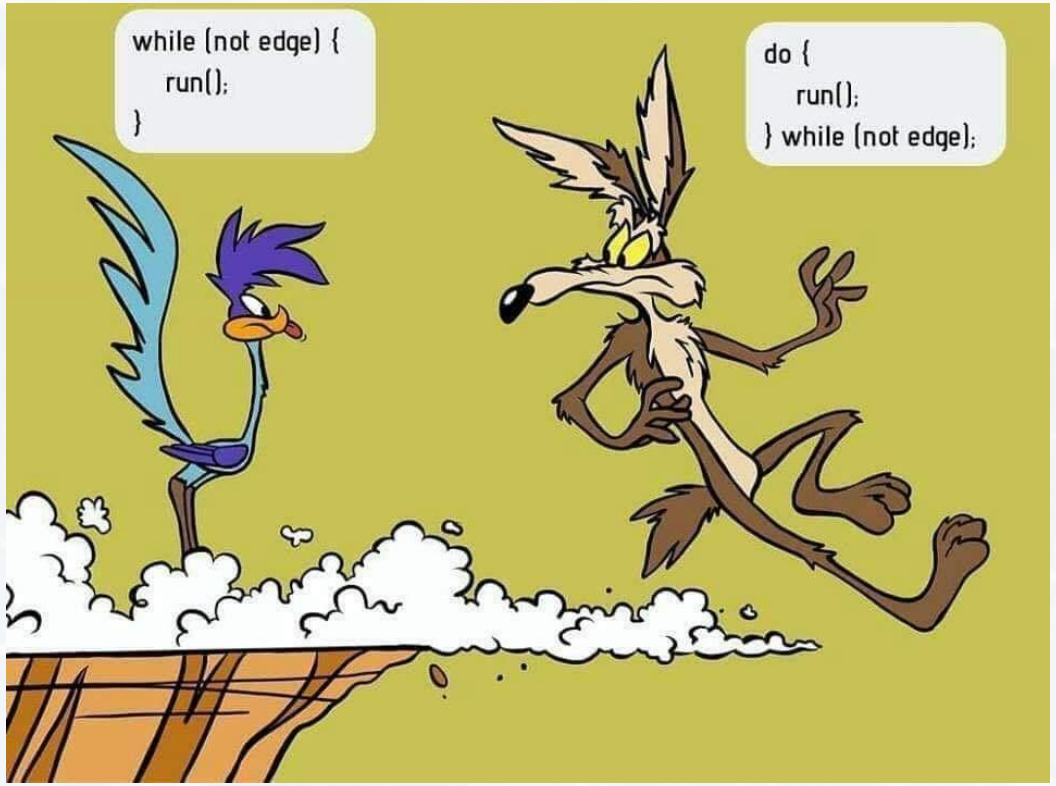
two conditions are true → **not** exclusive

```
for n in range(1,10):

    if n%2 == 0:
        print('even number: ' + str(n))

    elif n%3 == 0:
        print('multiple of 3: ' + str(n))

    elif n%5 == 0:
        print('multiple of 5: ' + str(n))
```




```
even number: 2
multiple of 3: 3
even number: 4
multiple of 5: 5
even number: 6
even number: 8
multiple of 3: 9
```

once a condition applies → runs statement **exclusively**

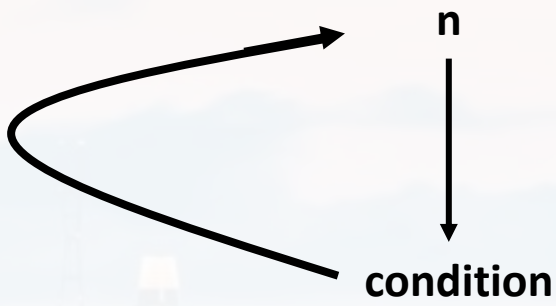


Outline

- General Idea and Structure
- `for` Loops and Comprehension
- `if`, `else` and `elif`
- **`while`**
- `break`, `continue` and `pass`

Control Structures

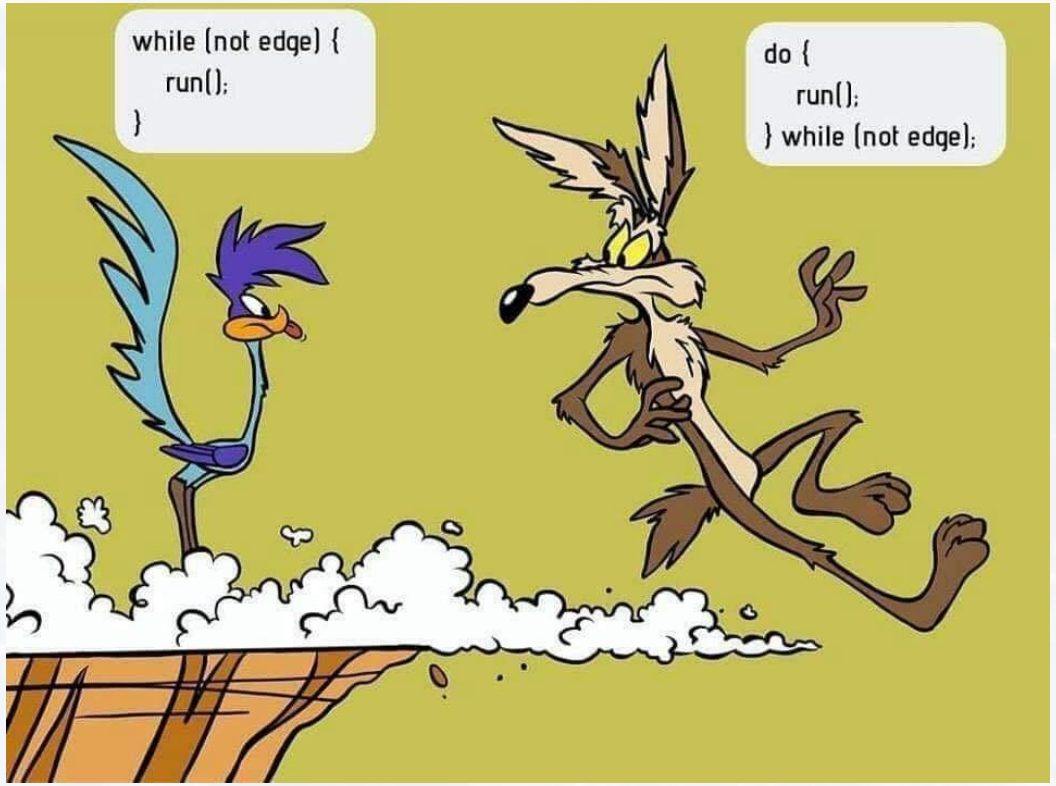

```
n = 0

while n < 10:

    n += 1
    print(n)
```



runs until condition is not true

```
1
2
3
4
5
6
7
8
9
10
```

note: make sure, that condition will be false at some point → may run infinitely → logical error



Outline

- General Idea and Structure
- `for` Loops and Comprehension
- `if`, `else` and `elif`
- `while`
- **`break`, `continue` and `pass`**

checking, if integer N>3 is a prime number

```
N = 40

for n in range(3,N):

    result = N%n

    if result == 0:
        print('not prime')
```

```
not prime
not prime
not prime
not prime
not prime
not prime
```

it is sufficient to know for the first time if *N* is not prime

→ don't need to run the entire loop

checking, if integer N>3 is a prime number

```
N = 40

for n in range(3,N):

    result = N%n

    if result == 0:
        print('not prime')

        break
```

interrupts loop immediately, if condition is true

```
not prime
```

checking, if integer N>3 is a prime number
Now we also want the code to print if **N is prime** too

```
N = 40

for n in range(3,N):

    result = N%n

    if result == 0:
        print('not prime')

        break

    else:
        print('is prime')
```

```
is prime
not prime
```

now says *'is prime'*, **each time** *N* is not dividable without remainder

checking, if integer N>3 is a prime number
Now we also want the code to print if **N is prime** too

```
N = 40

for n in range(3,N):

    result = N%n

    if result == 0:
        print('not prime')

        break

    else:
        if n < N-1:
            continue

        print('is prime')
```

skips the current iteration

```
N = 39  not prime

N = 40  not prime

N = 41  is prime
```

cleverer: **avoiding half of the iterations** in the first place:

```
if N > 1:
        for n in range(2, (N//2)+1):
                if (N % n) == 0:
                        print(N, 'not prime')
                        break
        else:
                print(N, 'is prime')
else:
        print(N, 'is not prime')
```

```
7  N = 41
8
9  if N > 1:
10
11
12
13
```

Syntax error before entering a statement.

Maybe you want to run the code without having written a statement!

```
7  N = 41
8
9  if N > 1:
10     pass
```

`pass` is a null statement: it matches the required syntax, but doesn't do anything, just a placeholder

Thank you very much for your attention!