Introduction to Bioinformatics using Python

Lecture 4: Control statements and loops

Dr. Alexey Larionov

28 October 2024

www.cranfield.ac.uk

Lecture plan (learning outcomes)

At the end of this lecture, you will be able to:

- Identify different flow control statements in Python
- Apply *if – elif – else* conditional statements
- Understand *match – case* statement
- Use *for* and *while* loops in Python code

Execution flow control

Sequential commands

Conditional execution

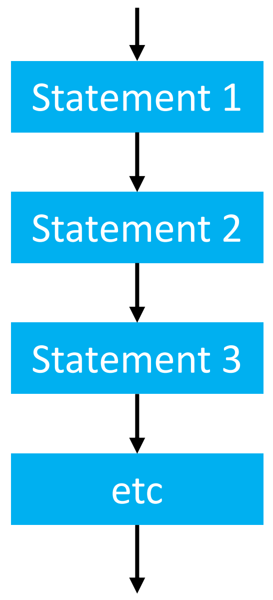
Loops

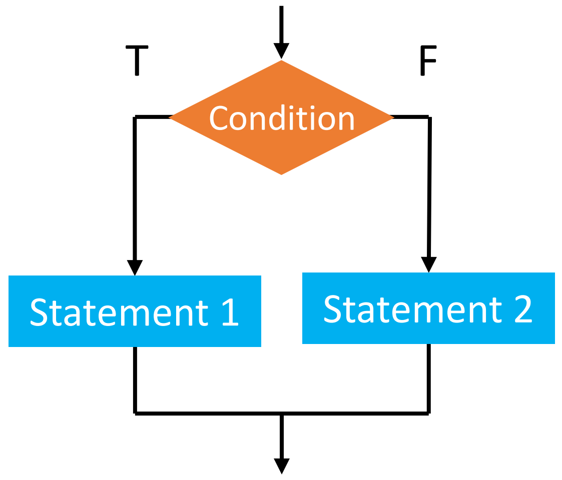
if - else

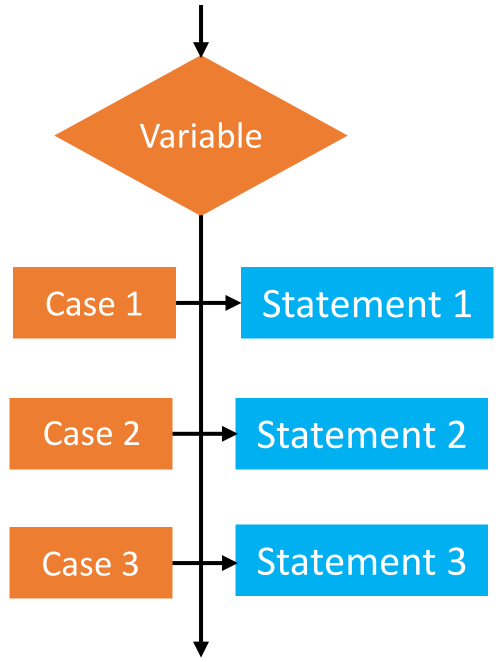
match - case

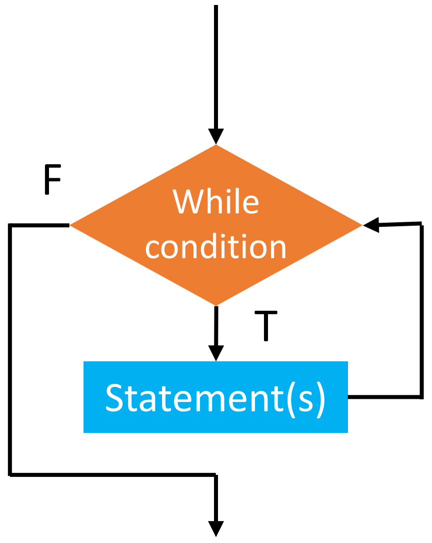
since Python 3.10 !

for / while

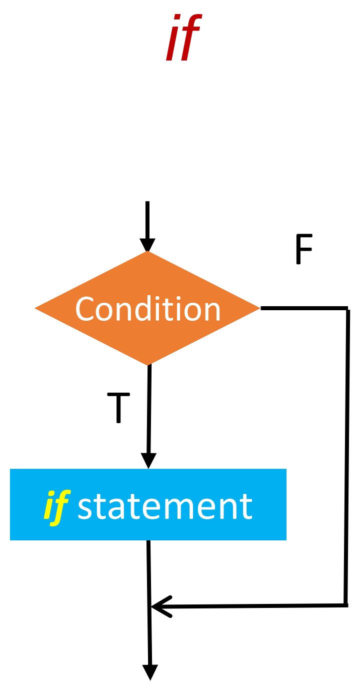


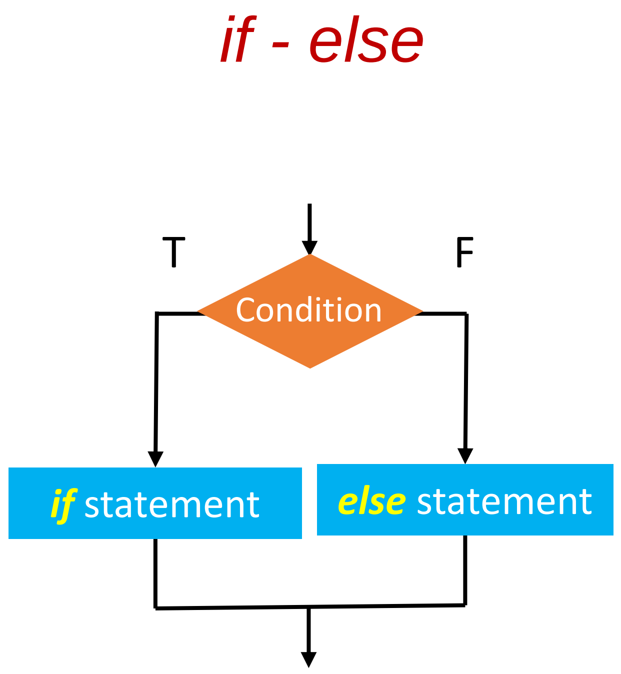


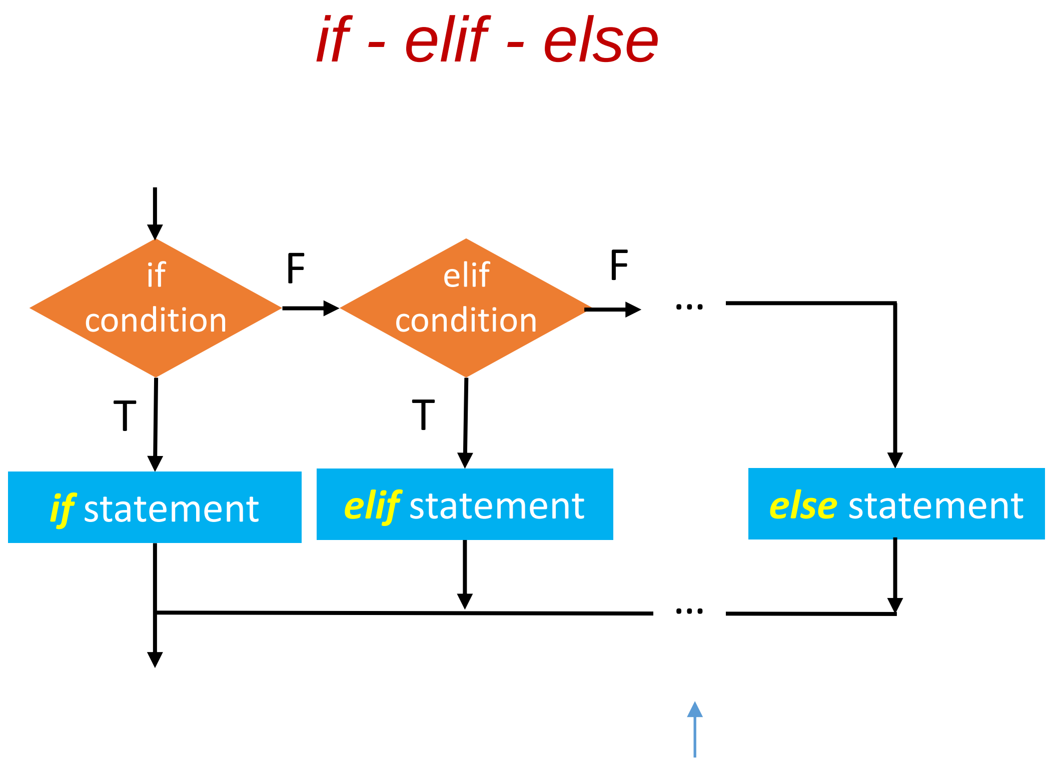




Different flavours of *if*







Multiple *elif* conditions can be added

The *if* statement

```
# Simple conditional statement

if condition :

       # do something
```

```
base = input("Input a nucleotide: ")


if base == 'A' :
       print("This is a valid base")
```

The *if* statement

```
# Simple conditional statement

if condition :
    # do something
```

Colon

Indent

Note the difference:
Equality check ==
Assignment =

Equality check is for strings and numbers

“is” may be used for more complicated **objects**

```
base = input("Input a nucleotide: ")

if base == 'A' :
       print("This is a valid base")
```

The *if* statement

```
# Simple conditional statement

if condition :
    # do something
```

Colon

Indent

Note the difference:
Equality check ==
Assignment =

Equality check is for strings and numbers

```
base = input("Input a nucleotide: ")

if(base == 'A'):
    print("This is a valid base")
```

"is" may be used for more complicated **objects**

Brackets here are OK, but they are not necessary: "syntactic sugar" for programmers who like using brackets

if & *if-else* statements

```
# Simple conditional statement

if condition :   Colon
Indent
          #do something
```

```
base = input("Input a nucleotide: ")


if base == 'A' :
       print("This is a valid base")
```

```
#One-alternative conditional statement
if condition :
          #This is the True branch
          #do something
else :
          #This is the False branch
          #do something else
```

```
base = input("Input a nucleotide: ")


if base == 'A' :
       print("This is a valid base")
else:
       print("This is not a valid base")
```

If base == ‘a’, will it be a valid base?   **NO**

if & *if-else* statements

What is wrong ?

```
print("Input a nucleotide:")
base = input()

if base == 'A' :
        print("This is a valid base: Adenine")
if base == 'T' :
        print("This is a valid base: Thymine")
if base == 'G' :
        print("This is a valid base: Guanine")
if base == 'C' :
        print("This is a valid base: Cytosine")
if base == 'U' :
        print("This is a valid base: Uracil")
else:
        print("This is not a valid base")
```

```
base = “A”

This is a valid base: Adenine
This is not a valid base
```

if & *if-else* statements

Nothing is wrong !

```
if(base == 'A'):
        print("This is a valid base: Adenine")
```

```
if(base == 'T'):
        print("This is a valid base: Thymine")
```

```
if(base == 'G'):
        print("This is a valid base: Guanine")
```

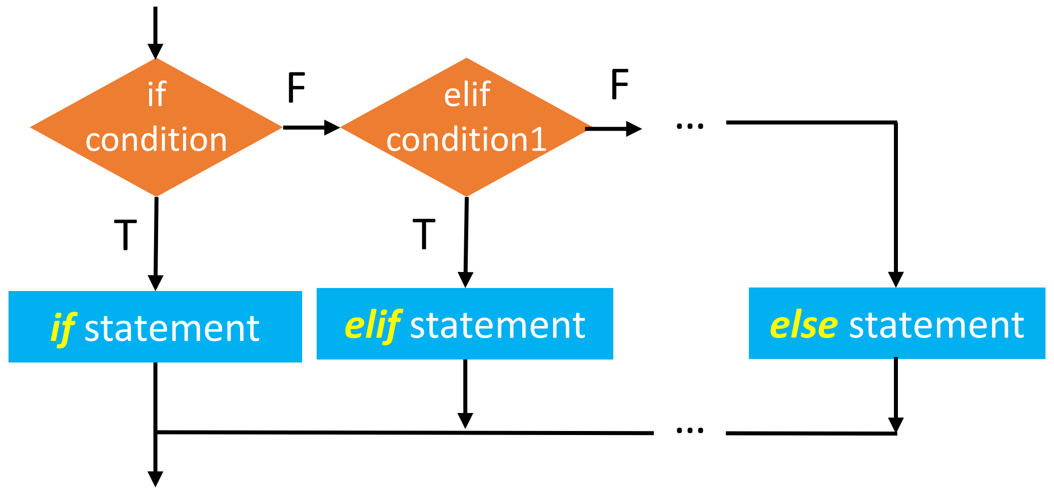
```
if(base == 'C'):
        print("This is a valid base: Cytosine")
```

```
if(base == 'U'):
        print("This is a valid base: Uracil")
else:
        print("This is not a valid base")
```

```
base = "A"
This is a valid base: Adenine
This is not a valid base
```

The *if - elif - else* statement

```
if condition :
       # This is branch 1
       # do something
elif condition1 :
       # This is branch 2
       # do something
elif condition2 :
       # This is branch 3
       # do something
else :
       # else branch
       # Executed if all other conditions are false
       # do something else
```



Multiple alternative conditions in the same statement

The *if - elif - else* statement

A single *if-elif-else* statement : exits after the first match!

```
print("Input a nucleotide:")
base = input()

if(base == 'A'):
        print("This is a valid base: Adenine")
elif(base == 'T'):
        print("This is a valid base: Thymine")
elif(base == 'G'):
        print("This is a valid base: Guanine")
elif(base == 'C'):
        print("This is a valid base: Cytosine")
elif(base == 'U'):
        print("This is a valid base: Uracil")
else:
        print("This is not a valid base")
```

```
base = "A"

This is a valid nucleobase: Adenine
```

See the difference?

```
base = input("Input a nucleotide: ")

if(base == 'A'):
        print("This is a valid base: Adenine")
if(base == 'T'):
        print("This is a valid base: Thymine")
if(base == 'G'):
        print("This is a valid base: Guanine")
if(base == 'C'):
        print("This is a valid base: Cytosine")
if(base == 'U'):
        print("This is a valid base: Uracil")
else:
        print("This is not a valid base")
```

```
base = "A"

This is a valid base: Adenine
This is not a valid base
```

```
base = input("Input a nucleotide: ")

if(base == 'A'):
        print("This is a valid base: Adenine")
elif(base == 'T'):
        print("This is a valid base: Thymine")
elif(base == 'G'):
        print("This is a valid base: Guanine")
elif(base == 'C'):
        print("This is a valid base: Cytosine")
elif(base == 'U'):
        print("This is a valid base: Uracil")
else:
        print("This is not a valid base")
```

```
base = "A"

This is a valid base: Adenine
```

Multiple conditions can be combined

All five conditions in one

```
base = input("Input a nucleobase: ")
if base=='A' or base=='T' or base=='G' or base=='C' or base=='U' :
    print("This is a valid base")
else:
    print("This is not a valid base")
```

Don't mix-up English AND and the logic AND !

```
If base == 'A' and base == 'T' and base == 'G' and base == 'C' and base == 'U'
```

Multiple conditions can be combined

This is also allowed:
If 1 < x < 10 :
…

All five conditions in one

```
base = input("Input a nucleobase: ")
if base=='A' or base=='T' or base=='G' or base=='C' or base=='U' :
    print("This is a valid base")
else:
    print("This is not a valid base")
```

Don’t mix-up English AND and the logic AND !

```
If base == 'A' and base == 'T' and base == 'G' and base == 'C' and base == 'U'
```

Ternary operator (*inline if* statement)

One of the Pythonic "Figures of speech"

```
variable = value if condition else other_value
```

This means:

variable = value if the condition holds; otherwise: other_value

```
valid_nucleotides = ['A','T','G','C',’U’]

base = input("Please enter a nucleobase: ")

check_base = "valid base" if base in valid_nucleotides else "invalid base"
```

Lecture plan (learning outcomes)

At the end of this lecture, you will be able to:

- ✓ Identify different flow control statements in Python
- ✓ Apply *if – elif – else* conditional statements
- Understand *match – case* statement
- Use *for* and *while* loops in Python code

The *match* – *case* statement

In Python < 3.10

```
match base:
         ^
SyntaxError: invalid syntax
```

```python
base = input("Input a nucleotide: ")

match base:
    case "A":
        print("This is a valid base: Adenine")
    case "T":
        print("This is a valid base: Thymine")
    case "G":
        print("This is a valid base: Guanine")
    case "C":
        print("This is a valid base: Cytosine")
    case "U":
        print("This is a valid base: Uracil")
    case other:
        print("This is not a valid base")
```

Execution stops at the first match

https://plainenglish.io/blog/how-to-use-the-match-statement-in-python-896fbbc79d0e

The *match – case* statement

```
base = input("Input a nucleotide: ")

match base:
    case "A":
        print("This is a valid base: Adenine")
    case "T":
        print("This is a valid base: Thymine")
    case "G":
        print("This is a valid base: Guanine")
    case "C":
        print("This is a valid base: Cytosine")
    case "U":
        print("This is a valid base: Uracil")
    case other:
        print("This is not a valid base")
```

Execution stops at the first match

Any word could be used in the last statement:

```
case other_base:
  print(...)
```

```
case _:
  print(...)
```

https://plainenglish.io/blog/how-to-use-the-match-statement-in-python-896fbbc79d0e

Lecture plan (learning outcomes)

At the end of this lecture, you will be able to:

✓ • Identify different flow control statements in Python

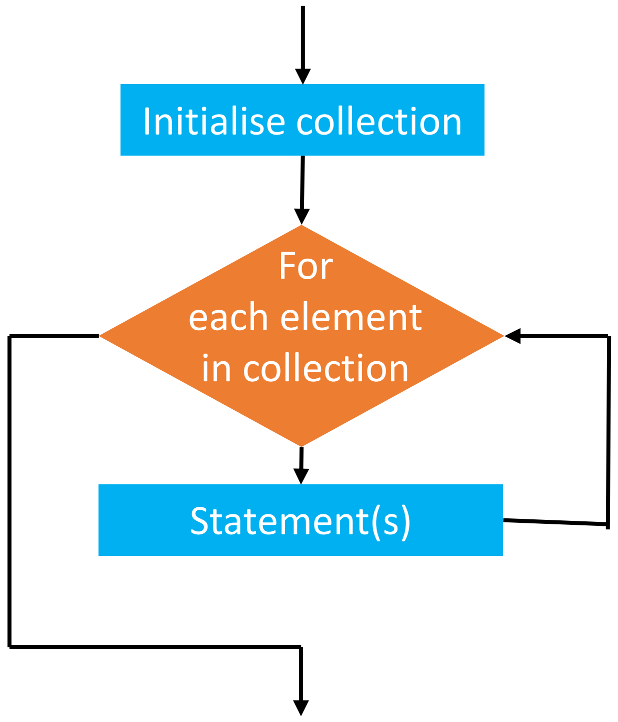
✓ • Apply *if – elif – else* conditional statements

✓ • Understand *match – case* statement

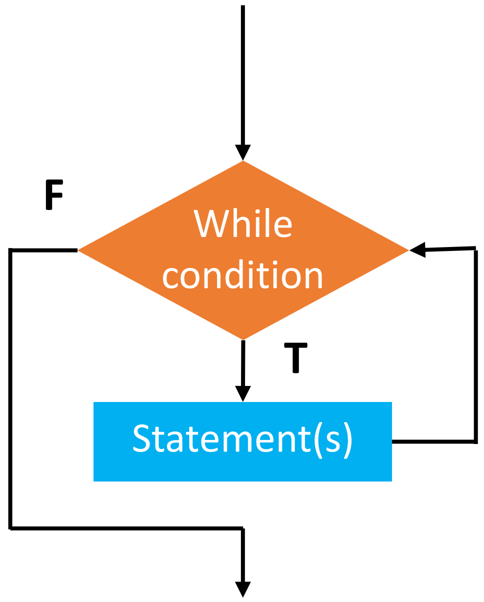
• Use *for* and *while* loops in Python code

Loops

for loop



while loop



The ***for*** loop is usually used for looping over ***iterable*** objects (**collections**, **ranges** etc)

Looping over collections

```
for item in collection :
    # Do something with the item
    # ...
```

str	*for* iterates over the characters
list	*for* iterates over the elements
tuple	*for* iterates over the elements
dict	*for* iterates over the keys

Looping over collections

str	*for* iterates over the characters
list	*for* iterates over the elements
tuple	*for* iterates over the elements
dict	*for* iterates over the keys

```
a_string = "Hello world again!"
a_list = [ "Hello", "world", "again!" ]
a_tuple = ( "Hello", "world", "again!" )
a_dictionary = {"w_1":"Hello", "w_2":"world", "w_3":"again!" }
```

```
for character in a_string:
    print(character)
```

```
H
e
l
…
```

```
for item in a_list:
    print(item)
```

```
Hello
world
again!
```

```
for item in a_tuple:
    print(item)
```

```
Hello
world
again!
```

```
for item in a_dictionary:
    print(item)
```

```
w_1
w_2
w_3
```

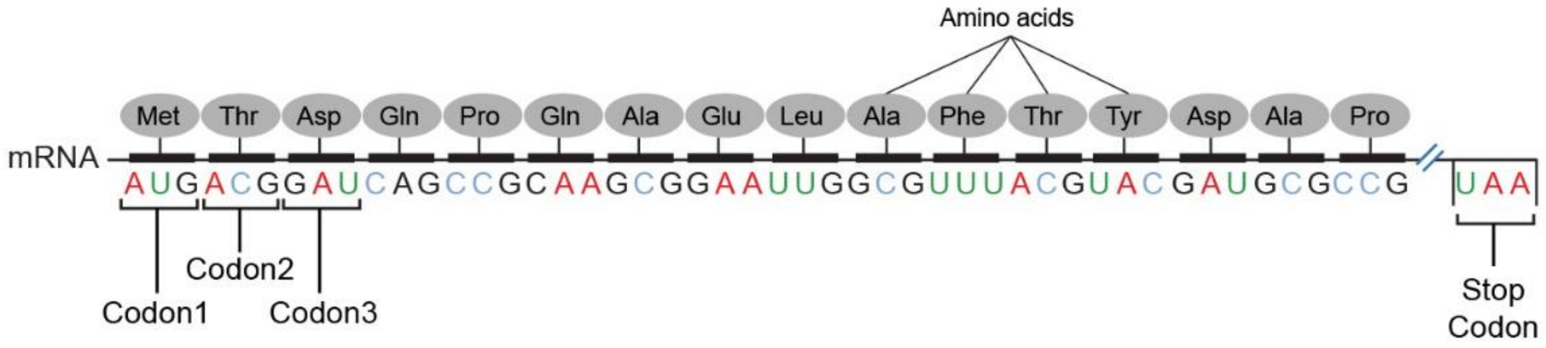
Looping over a range

No need to initialize and increment counter ☺

```
for index in range(start, end, step):
        # do something with index
        # ...
```

By default, `step = 1`

Translating RNA to Protein:



```
codons = ["UUC", "UCA", "UAA"]
for i in range(0, len(codons)):
        if( codons[i] == "UAA" ):
                print(f"{i+1}: Stop codon!")
        else:
                print(f"{i+1}: Amino acid codon")
```

Output:

```
1: Amino acid codon
2: Amino acid codon
3: Stop codon!
```

enumerate()

Pythonic way to code loops with counters

enumerate() is more Pythonic than looping over a range of indices ☺

```
for i, element in enumerate(collection, start):
        # do something with i and element
        # ...
```

By default, `start = 0`

```
codons = ["UUC", "UCA", "UAA"]

for i, codon in enumerate(codons, start=1):
        if( codon == "UAA" ):
                print(f"{i}: Stop codon!")
        else:
                print(f"{i}: Amino acid codon")
```

Output:

```
1: Amino acid codon
2: Amino acid codon
3: Stop codon!
```

https://realpython.com/python-enumerate

Break and Continue

```
          0         1         2         3
codons = ["UUC", "UCA", "UGA", "UUG"]
```

- The **break** command takes the program outside of the loop

```
for i in range(0, len(codons)):
      if( codons[i] == "UGA"):
            break
      print(f"{i} Protein codon")
```

Output

```
0 Protein codon
1 Protein codon
```

Break and Continue

```
             0         1          2          3
codons = ["UUC", "UCA", "UGA", "UUG"]
```

- **break** takes the program outside of the loop

```
for i, codon in enumerate(codons):
    if( codon == "UGA" ):
        break
    print(f"{i} : {codon}")
```

Output

```
0 : UUC
1 : UCA
```

- **continue** skips the rest of current iteration (proceeding to the next iteration)

```
for i, codon in enumerate(codons):
    if( codon == "UGA" ):
        continue
    print(f"{i} : {codon}")
```

Output

```
0 : UUC
1 : UCA
3 : UUG
```

While loop

While loop executes code while a specific condition holds true

```
while condition :
    # do something that
    # updates the value of condition
```

```
i = 0

while i < 22:
  i = i + 1
  print(f"Chromosome: chr{str(i)}")
```

The looped block of code is defined by indentation

Output:

```
Chromosome: chr1
Chromosome: chr2
…
Chromosome: chr21
Chromosome: chr22
```

The block of code should update the condition,
otherwise, the loop will continue indefinitely and never exit
(unless there is a ***break*** statement inside the block :)

Definite and Indefinite loops

Definite loop:

```
n = 1
stop = int(input())
while n <= stop:
      print(n)
      n += 1
```

Indefinite loop:

```
done = False

while (not done):
      entry = int(input())
      if entry == 42:
            done = True
      else:
            print(entry)
```

Brackets here are OK,
but they are not necessary:
"syntactic sugar" for programmers
who like using brackets

It's OK to have "indefinite" *while* loops:
a program may loop until a certain event happens

Of course, you cannot have indefinite *for* loop ☺

Nested loops

Loops within loops …

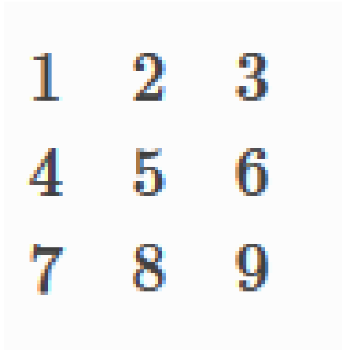
- Nested *for* loop:

```
for i in collection:
    for j in another_collection:
        # do some stuff with i and j
```

Printing a 3 x 3 matrix using Python's list of lists:

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
for row in matrix:
      line = ""
      for element in row:
            line = line + str(element) + " "
      print(line)
```

Output



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

Nested loops

Loops within loops …

- Combining **for** and **while** loops:

```
for i in (3,5,9) :
    j = 1
    output = ""
    while j <= i :
        output = str(j) + " " + output
        j += 1
        print(output)
```

Output:

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

Lecture plan (learning outcomes)

DONE

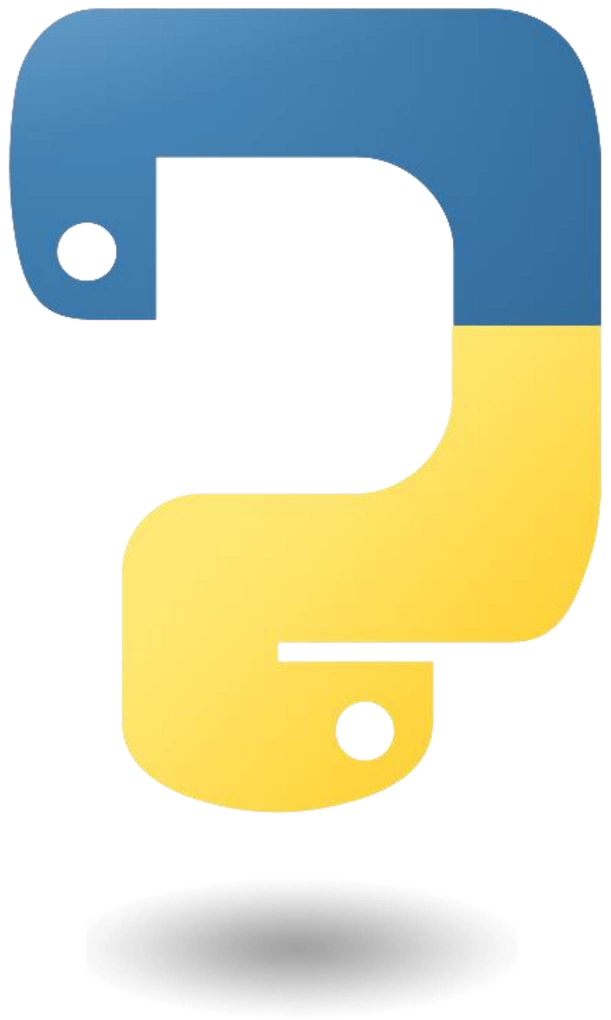
At the end of this lecture, you will be able to:

✓ • Identify different flow control statements in Python

✓ • Apply *if – elif – else* conditional statements

✓ • Understand *match – case* statement

✓ • Use *for* and *while* loops in Python code



Questions