UNIVERSITÉ DE LORRAINE

Institut des sciences du Digital Management & Cognition
COMPOSANTE DE L'UNIVERSITÉ DE LORRAINE

Programming

2- Strings, Lists, Control flow

Those slides will be available on Arche

Introduction

Gaël Guibon - gael.guibon@univ-lorraine.fr

Labs with Nasser-Eddine Monir (PhD Student - Loria - Multispeech)

Evaluation

70% exam

30% based on labs: For most labs (not the first one), you will have until the **day before the next lab session** to send me your solution.

Labs are not graded!

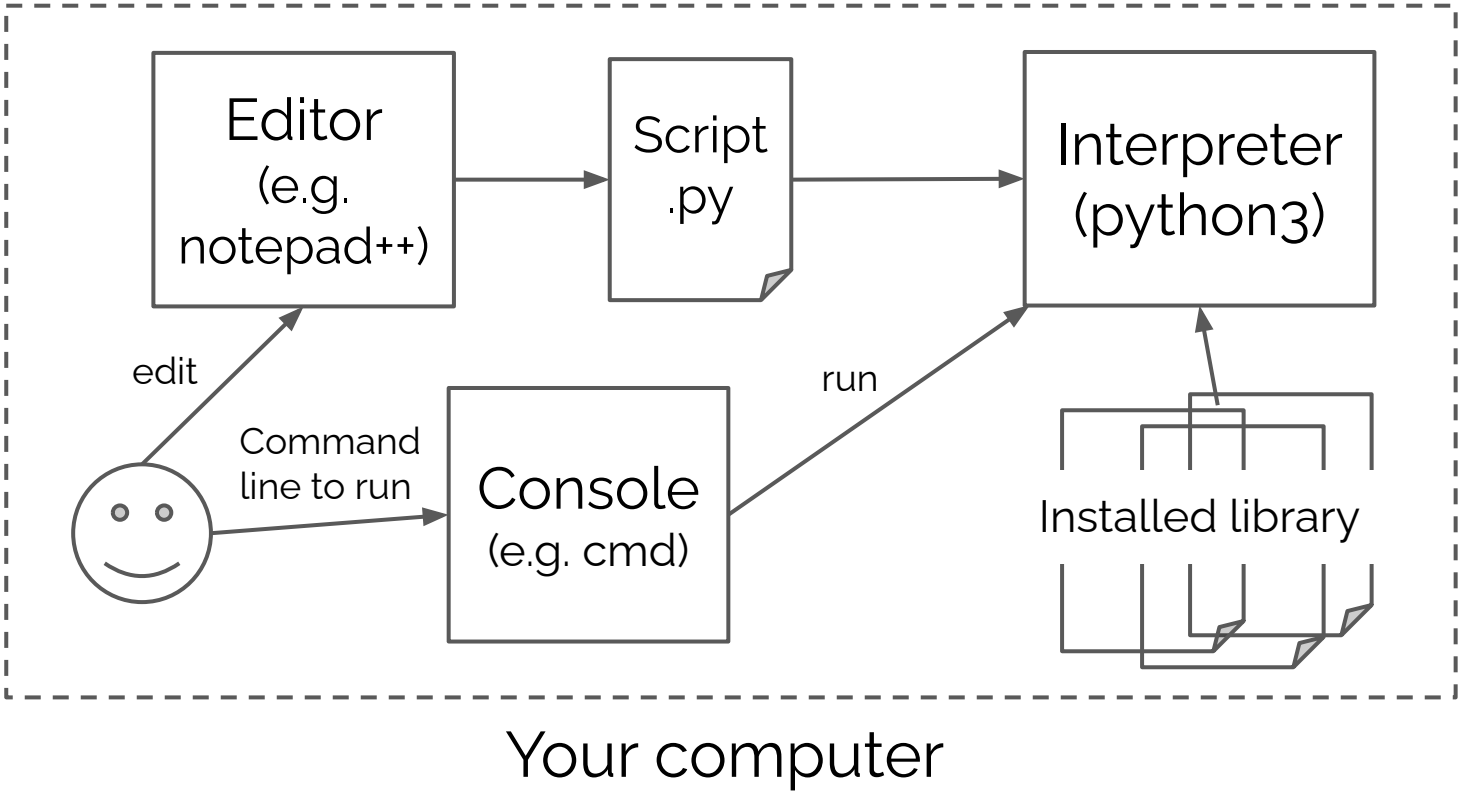
Content inspired from Mathieu d'Aquin

TODAY

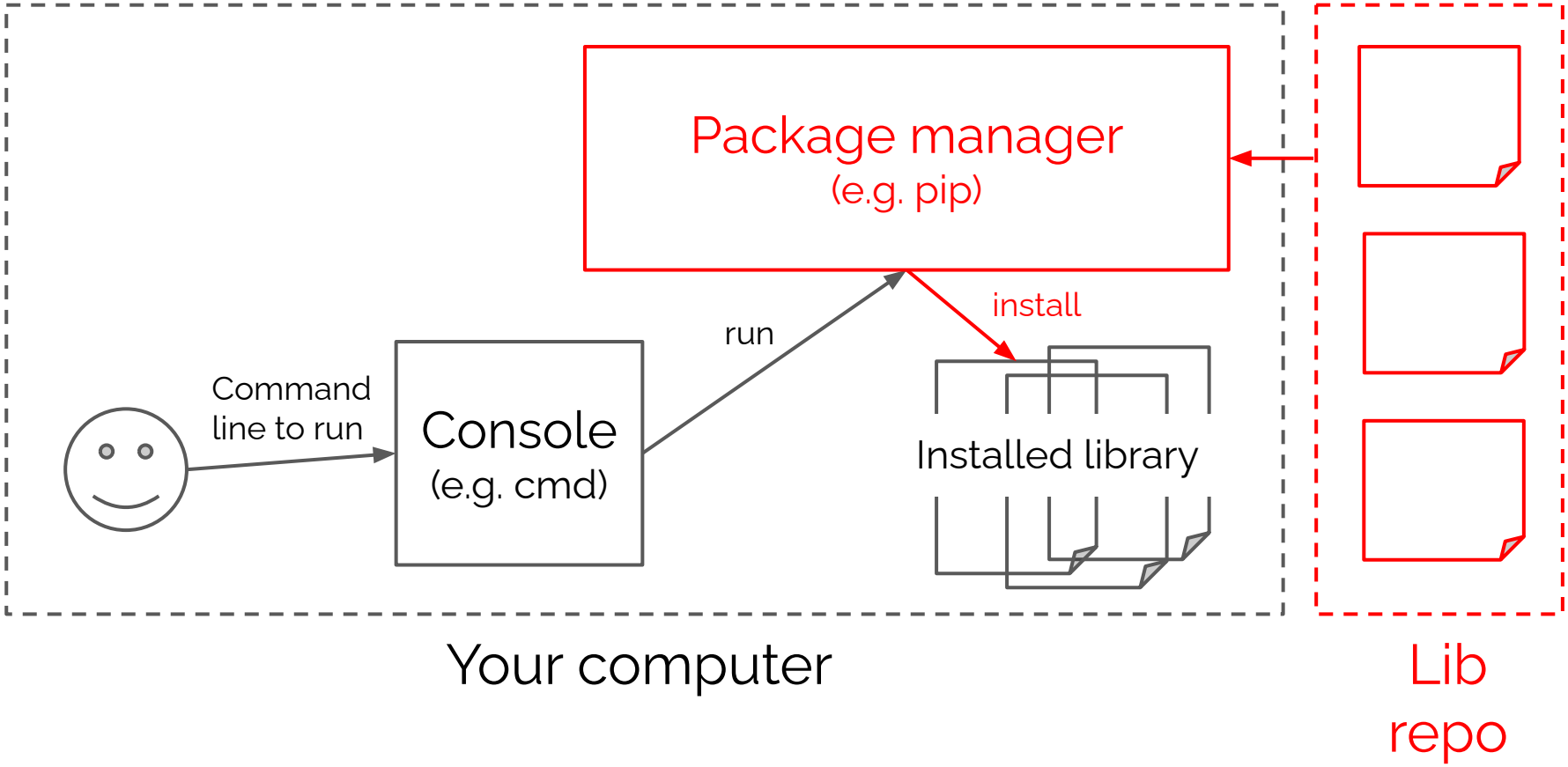
- Some additional points on topics from last time
  - ID, Input, Variable Identity
- Strings again
- Control Flow
- Iterations
- Lists & Loops

Python Environments

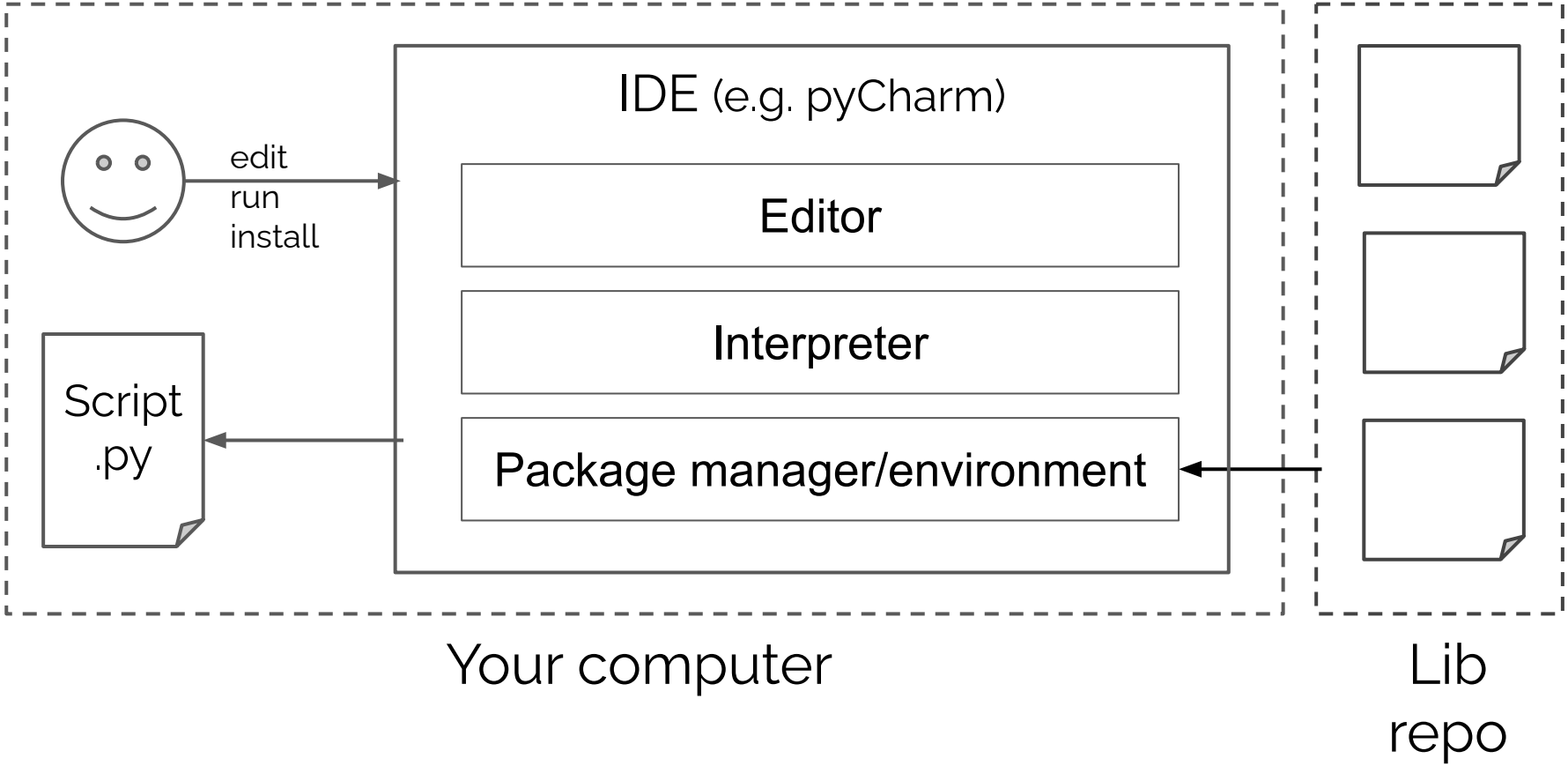
The very basic: Interpreter + editor



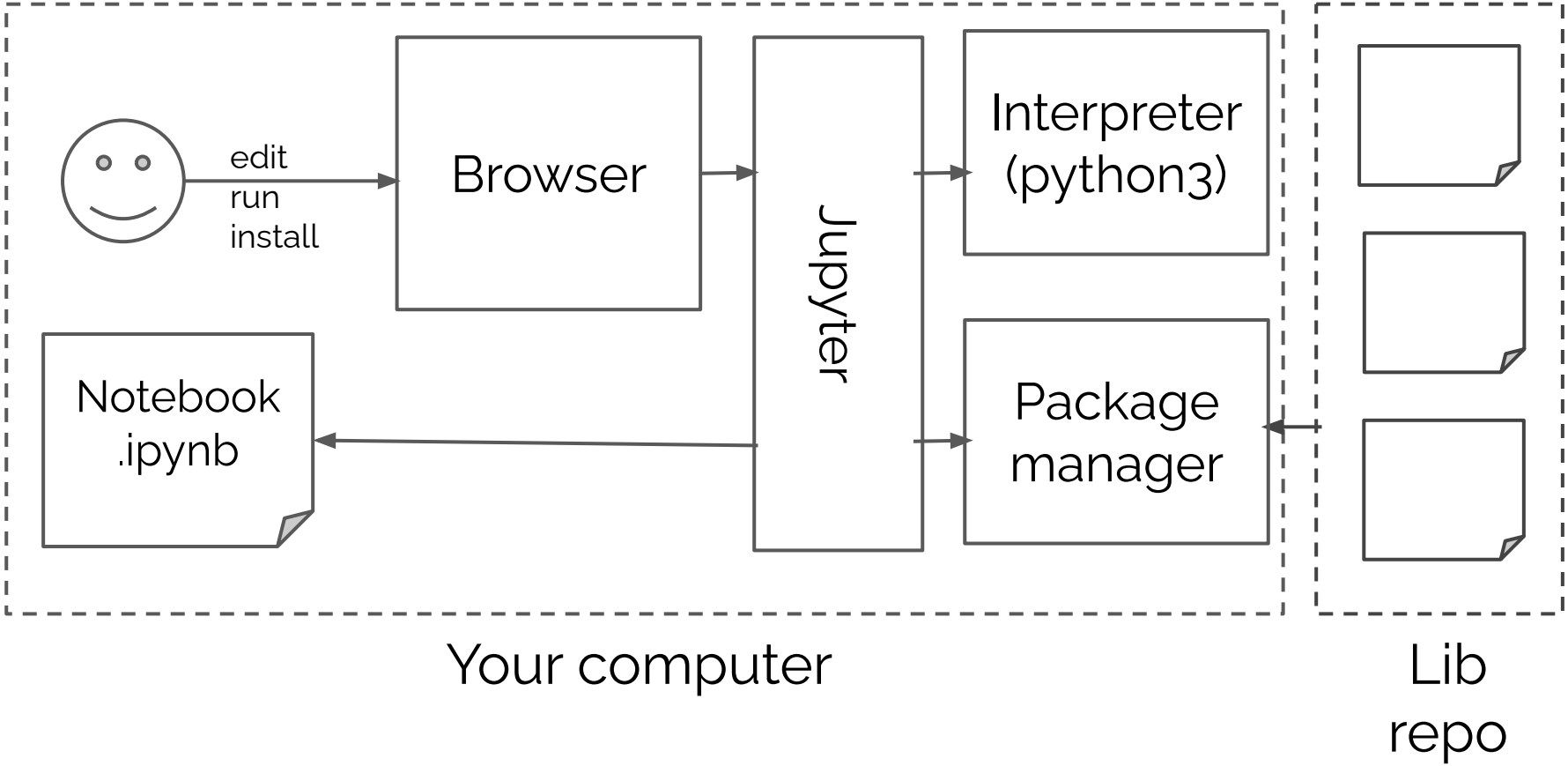
The very basic: Interpreter + editor



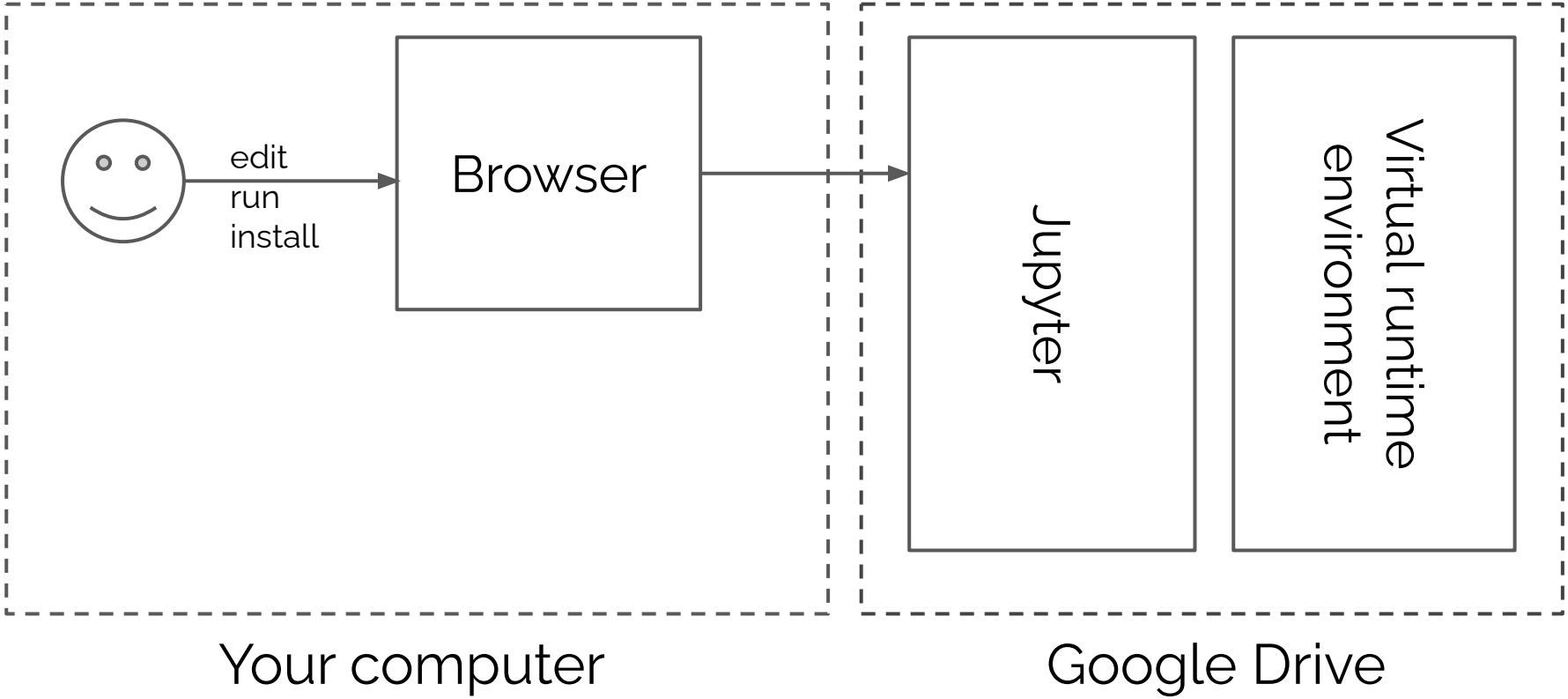
The less basic: IDE



The nice one: Jupyter Lab/Notebook



The cloudy one: Google Colab



More on Strings

More on Strings (str)

Strings (str) in python are sequences of characters. They can be declared using single or double quotes:

```
x1 = "I'm a string"
x2 = 'I am a “better” string'
```

Common operations on strings:

- `x+y` - concatenation (“bob”+’ is the best’ → “bob is the best”)
- `x[y]` - get the character at index y in the string x (x=”bob”, x[1] == “o”)
- `len(x)` - length of the string x (x=”bob”, len(x) == 3)

Multi-line Strings

Using three double or single quotes to start and finish the string, it is possible to write a string over multiple lines. For example,

```
'''Hello,
I am here, typing a multiline string,
which happens to be very convenient.'''
```

Is exactly the same as:

```
'''Hello,\nI am here, typing a multiline string,\nwhich happens to be very convenient.'''
```

Knowing that \n is the escape character for “newline”.

A parenthesis on escape characters

Some special characters can be included in strings using \ to indicate that it is an escape character, including

- \' Single quote (')
- \" Double quote (")
- \a ASCII Bell (BEL)
- \b ASCII Backspace (BS)
- \f ASCII Formfeed (FF)
- \n ASCII Linefeed (LF)
- \r ASCII Carriage Return (CR)
- \t ASCII Horizontal Tab (TAB)
- \v ASCII Vertical Tab (VT)
- \ooo ASCII character with octal value ooo
- \xhh... ASCII character with hex value hh...

and of course

- \\ Backslash \

A parenthesis on escape characters

\r means carriage returns such as in this funny video:

https://youtu.be/88LMBkCmnoQ?feature=shared

A parenthesis on escape characters

Unicode escape sequences:

- \N{name} Character named name in the Unicode database
- \uxxxx Character with 16-bit hex value xxxx
- \Uxxxxxxxx Character with 32-bit hex value xxxxxxxx

String prefixes

There can be prefixes in front on strings that affect the way python interpret the string.

In the example program from last time:

```
name = "bob"
version = 3.5
print(f"Hello, my name is {name}, version {version}")
```

String prefixes

There can be prefixes in front on strings that affect the way python interpret the string.

In the example program from last time:

```
name = "bob"
version = 3.5
print(f"Hello, my name is {name}, version {version}")
```

String prefixes

none	Unicode string, with escape characters (byte string in python2)
b	byte string
u	unicode string
r	raw string (escape character are not escaped)
rb,br	raw byte string
f	formatted string (can be combined with u and b)

Other interesting operations on strings

Other useful operations on strings:

- **Repeat concatenation:** `"AB"*3 # >>> 'ABABAB'`
- **`Lower case:`** `"AbaAbAbbbA".lower() # >>> 'abaababbba'`
- **`Upper case:`** `"AbaAbAbbbA".upper() # >>> 'ABAABABBBA'`
- **`Find a substring:`** `"Gaël".find("ie") # >>> 4`
- **`Replace:`** `"Ireland".replace("re", "ce")#>>> 'Iceland'`

A Take on Variable Identities

Types in python are inferred

Other programming languages require variables to be explicitly typed. For example, in Java:

```
String message = “hello”;
int niceNumber = 123;
```

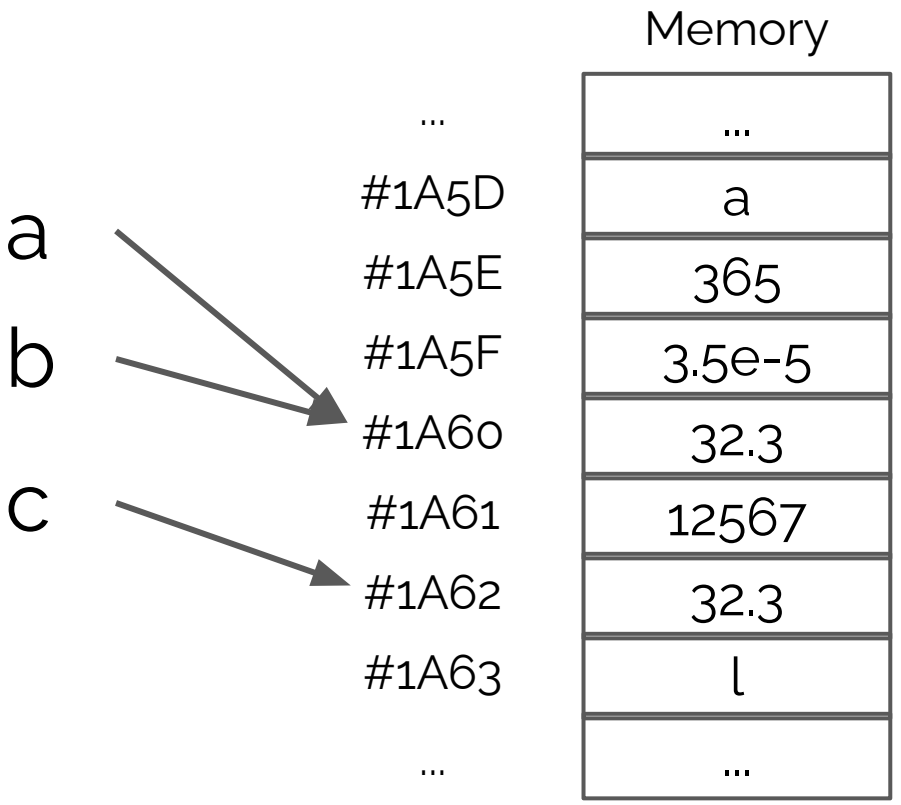
Python does not require this: The type of a variable is dynamically inferred from its value, which also means that it can change.

```
a = 3.5
print(type(a)) # >>> float
a = 3
print(type(a)) # >>> int
a = "bob"
print(type(a)) # >>> str
```

is vs ==

== tests the values, is test the "pointer"

```
a = ["apple", "banana", "cherry"]
b = ["apple", "banana", "cherry"]
print(a is b)
print(a == b)
a = b
print(a is b)
print(a == b)
```



More on input()

Input in python

You have seen in in the lab that there is one function to get input from the user: `input()`

It returns a string that corresponds to what the user typed up to the point when they hit enter.

Optionally, you can include a prompt as a parameter:

```
input("your string here")
```

Input/output in python

```
>>> name = input()
Gaël
>>> print(name)
Gaël
>>> job = input(f"Hi {name}, what's your job?\n")
Hi Gaël, what's your job?
Mage
>>> prof = input("Who is your professor?\n")
Who is your professor?
Dumbledore
>>> print(f"name: {name}, job: {job}, prof: {prof}")
name: Gaël, job: Mage, prof: Dumbledore
```

More on print()

Print: Writing on the screen/console/output

Print write a string on the screen on one line (by default).

```
print("hello my friend.") # >>> hello my friend.
```

It can take multiple parameters and will, by default, concatenate them with spaces

```
pi=math.pi
print("The value of pi is", pi,"!")
```

But you can change the separator and end if you like:

```
print("The value of pi is", pi, sep=": ", end="!\n")
```

Control Flow

Control Flow

Control flow is what decides in which order instructions are executed in a program.

There are generally three ways in which that can happen:

- **Sequential**: Instructions are executed in the order they appear
- **Branching**: Choosing which instructions to follow given a condition
- **Iteration**: Looping through instructions while a certain condition is met

Control Flow

Control flow is what decides in which order instructions are executed in a program.

There are generally three ways in which that can happen:

- **Sequential**: Instructions are executed in the order they appear
- **Branching**: Choosing which instructions to follow given a condition
- **Iteration**: Looping through instructions while a certain condition is met

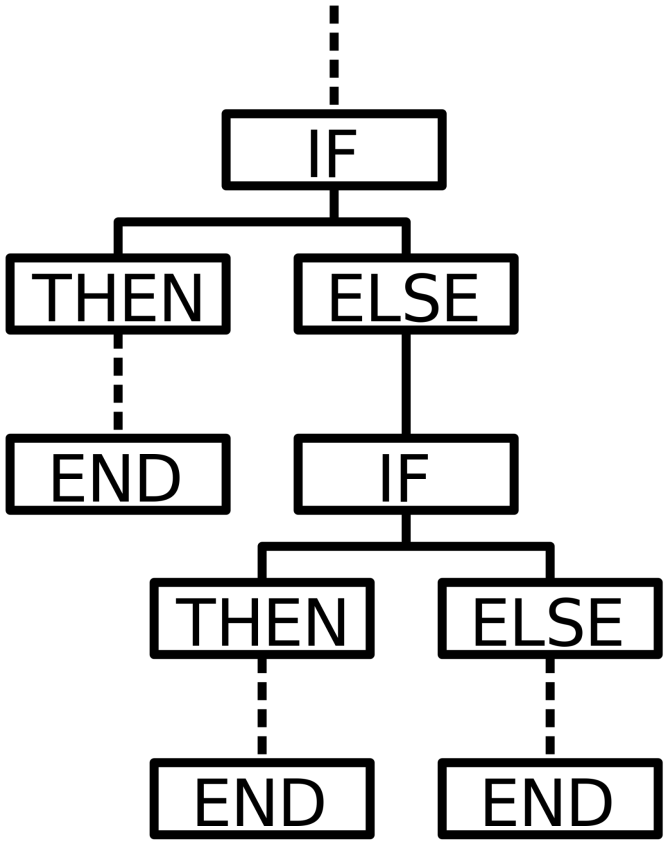
```
1  Do Instruction 1
2  Do Instruction 2
3  etc.
```

Control Flow

Control flow is what decides in which order instructions are executed in a program.

There are generally three ways in which that can happen:

- **Sequential**: Instructions are executed in the order they appear
- **Branching**: Choosing which instructions to follow given a condition
- **Iteration**: Looping through instructions while a certain condition is met

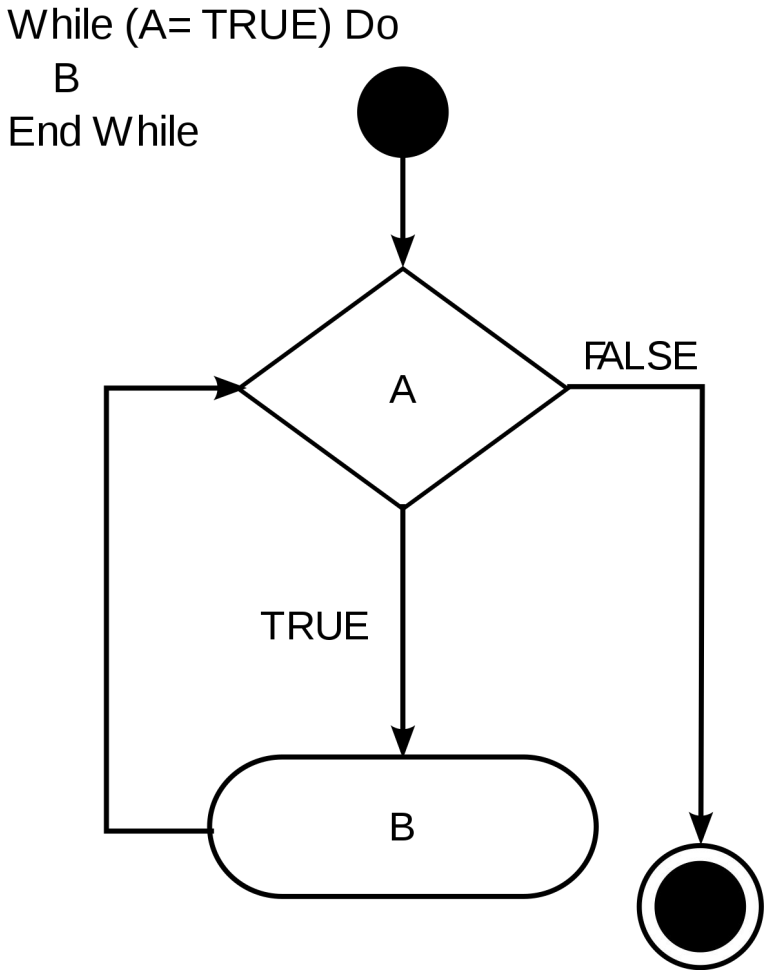


Control Flow

Control flow is what decides in which order instructions are executed in a program.

There are generally three ways in which that can happen:

- **Sequential**: Instructions are executed in the order they appear
- **Branching**: Choosing which instructions to follow given a condition
- **Iteration**: Looping through instructions while a certain condition is met

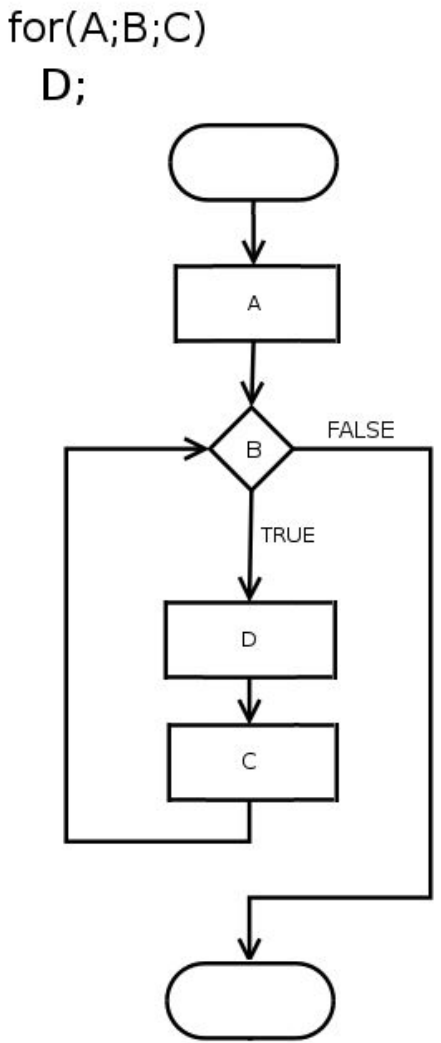


Control Flow

Control flow is what decides in which order instructions are executed in a program.

There are generally three ways in which that can happen:

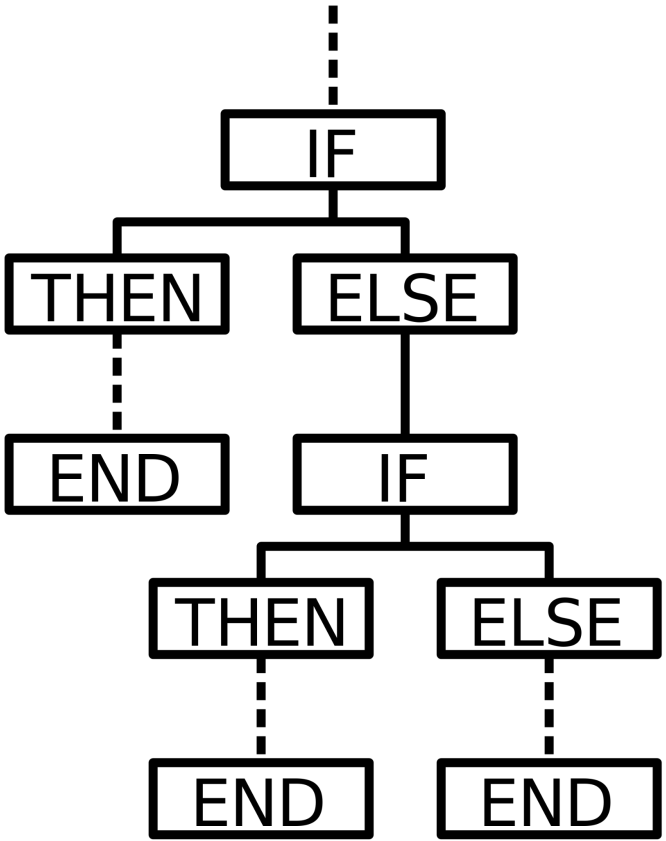
- **Sequential**: Instructions are executed in the order they appear
- **Branching**: Choosing which instructions to follow given a condition
- **Iteration**: Looping through instructions while a certain condition is met



Note on Syntax

Do not forget the colon! :

```
1 if condition:
2     instruction
3 elif condition:
4     instruction
5 else:
6     instruction
```



Branching / conditional

```
age = int(input("your age?\n"))
if age > 40:
    lifeExp = 82
    print("your old!")
    timeR = lifeExp - age
    print(f"you are likely to die in the next {timeR} years")
elif age > 30:
    print("hey, you are a full grown adult now, well done!")
    yearSM = age - 18
    print(f"you have reached majority for {yearSM} years :)")
else:
    print("Cool, a young one :)")
    days = age * 365
    print(f"Only a bit more than {days} days since your were born")
```

Branching / conditional

```
age = int(input("your age?\n"))
if age > 40:
    lifeExp = 82
    print("your old!")
    timeR = lifeExp - age
    print(f"you are likely to die in the next {timeR} years")
elif age > 30:
    print("hey, you are a full grown adult now, well done!")
    yearSM = age - 18
    print(f"you have reached majority for {yearSM} years :)")
else:
    print("Cool, a young one :)")
    days = age * 365
    print(f"Only a bit more than {days} days since your were born")
```

Condition (followed by :)

Branching / conditional

Block of code executed if condition is True

```
age = int(input("your age?\n"))
if age > 40:
    lifeExp = 82
    print("your old!")
    timeR = lifeExp - age
    print(f"you are likely to die in the next {timeR} years")
elif age > 30:
    print("hey, you are a full grown adult now, well done!")
    yearSM = age - 18
    print(f"you have reached majority for {yearSM} years :)")
else:
    print("Cool, a young one :)")
    days = age * 365
    print(f"Only a bit more than {days} days since your were born")
```

Branching / conditional

```
age = int(input("your age?\n"))
if age > 40:
    lifeExp = 82
    print("your old!")
    timeR = lifeExp - age
    print(f"you are likely to die in the next {timeR} years")
elif age > 30:
    print("hey, you are a full grown adult now, well done!")
    yearSM = age - 18
    print(f"you have reached majority for {yearSM} years :)")
else:
    print("Cool, a young one :)")
    days = age * 365
    print(f"Only a bit more than {days} days since your were born")
```

else if - new condition if the first one is False

optional - multiple

Branching / conditional

```
age = int(input("your age?\n"))
if age > 40:
    lifeExp = 82
    print("your old!")
    timeR = lifeExp - age
    print(f"you are likely to die in the next {timeR} years")
elif age > 30:
    print("hey, you are a full grown adult now, well done!")
    yearSM = age - 18
    print(f"you have reached majority for {yearSM} years :)")
else:
    print("Cool, a young one :)")
    days = age * 365
    print(f"Only a bit more than {days} days since your were born")
```

Block of code executed if first condition is False but second one is True

Branching / conditional

```
age = int(input("your age?\n"))
if age > 40:
    lifeExp = 82
    print("your old!")
    timeR = lifeExp - age
    print(f"you are likely to die in the next {timeR} years")
elif age > 30:
    print("hey, you are a full grown adult now, well done!")
    yearSM = age - 18
    print(f"you have reached majority for {yearSM} years :)")
else:
    print("Cool, a young one :)")
    days = age * 365
    print(f"Only a bit more than {days} days since your were born")
```

else - for when all conditions are False

Branching / conditional

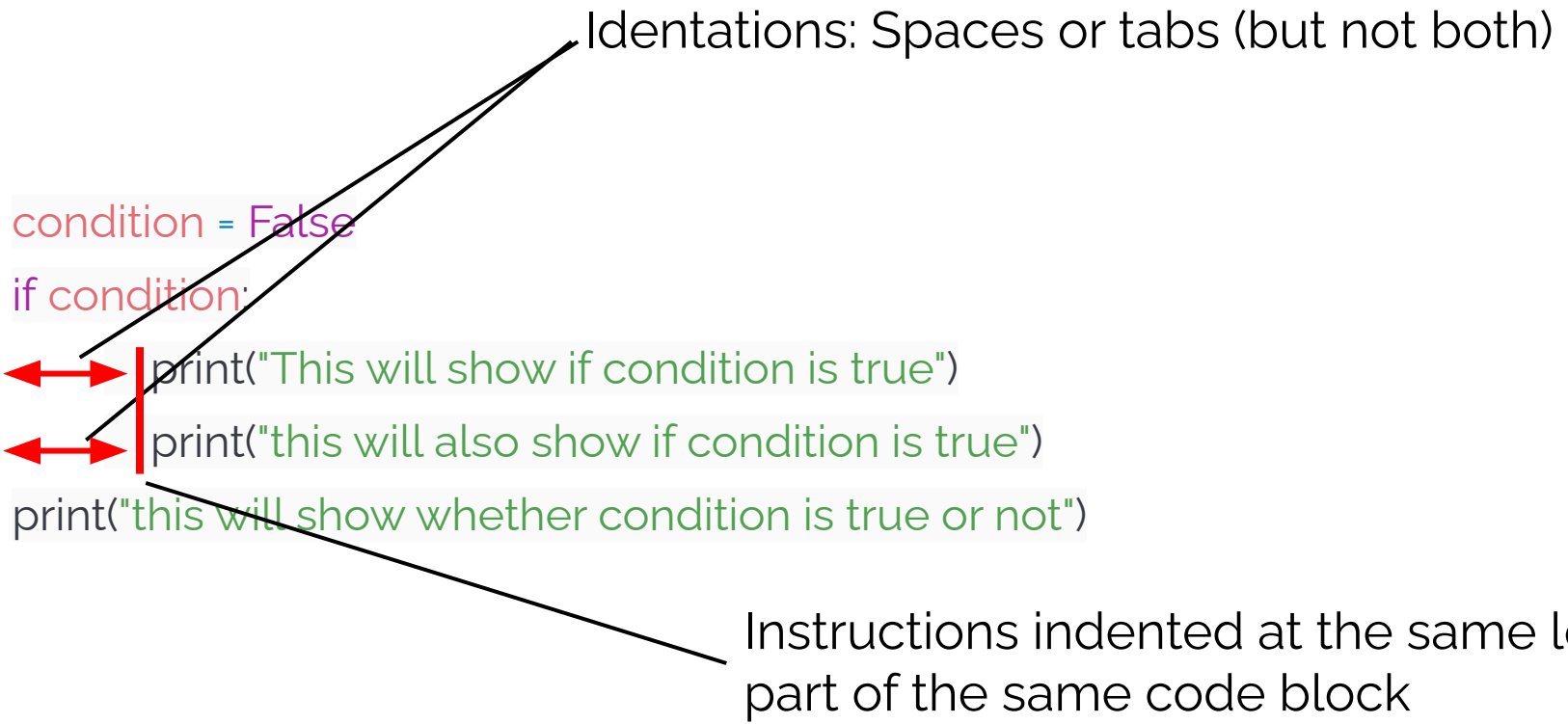
```
age = int(input("your age?\n"))
if age > 40:
    lifeExp = 82
    print("your old!")
    timeR = lifeExp - age
    print(f"you are likely to die in the next {timeR} years")
elif age > 30:
    print("hey, you are a full grown adult now, well done!")
    yearSM = age - 18
    print(f"you have reached majority for {yearSM} years :)")
else:
    print("Cool, a young one :)")
    days = age * 365
    print(f"Only a bit more than {days} days since your were born")
```

Block of code executed if both conditions are False **optional**

A parenthesis on indentation

```
condition = False
if condition:
        print("This will show if condition is true")
        print("this will also show if condition is true")
print("this will show whether condition is true or not")
```

A parenthesis on indentation



A parenthesis on indentation

```
condition = False
if condition:
print("This will fail: if requires a new block")
        print("this will also show if condition is true")
print("this will show whether condition is true or not")
```

A parenthesis on indentation

```
condition = False
if condition:
        print("This will show if condition is true")
   print("this will fail: it cannot be in between blocks")
print("this will show whether condition is true or not")
```

A parenthesis on indentation

```
condition = False
if condition:
    print("This will show if condition is true")
      print("this will fail: why starting a new block?")
print("this will show whether condition is true or not")
```

Blocks within blocks

This is equivalent to the previous code, but using two ifs, rather than elif.

```
age = int(input("your age?\n"))
if age > 40:
        lifeExp = 82
        print("your old!")
        timeR = lifeExp - age
        print(f"you are likely to die in the next {timeR} years")
else:
        if age > 30:
                print("hey, you are a full grown adult now, well done!")
                yearSM = age - 18
                print(f"you have reached majority for {yearSM} years :)")
        else:
                print("Cool, a young one :)")
                days = age * 365
                print(f"Only a bit more than {days} days since your were born")
```

Blocks within blocks

This is equivalent to the previous code, but using two ifs, rather than elif.

```
age = int(input("your age?\n"))
if age > 40:
        lifeExp = 82
        print("your old!")
        timeR = lifeExp - age
        print(f"you are likely to die in the next {timeR} years")
else:
        if age > 30:
                print("hey, you are a full grown adult now, well done!")
                yearSM = age - 18
                print(f"you have reached majority for {yearSM} years :)")
        else:
                print("Cool, a young one :)")
                days = age * 365
                print(f"Only a bit more than {days} days since your were born")
```

A parenthesis on something else

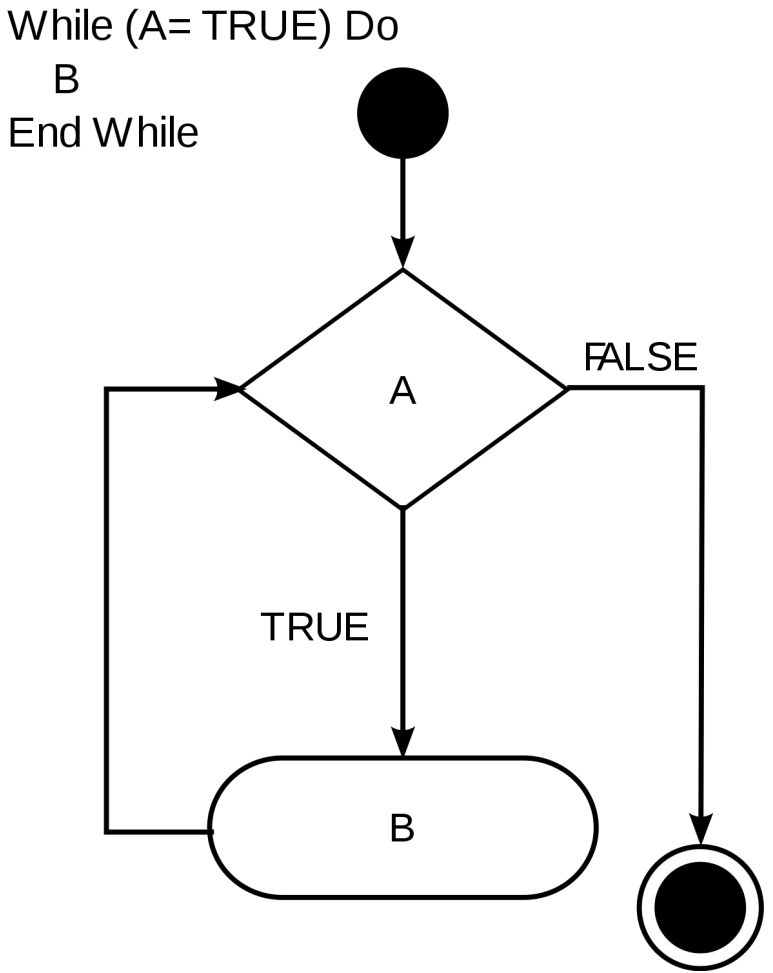
```
age = int(input("your age?\n"))
if age > 40:
        lifeExp = 82
        print("your old!")
        timeR = lifeExp - age
        print(f"you are likely to die in the next {timeR} years")
elif age > 30:
        print("hey, you are a full grown adult now, well done!")
        yearSM = age - 18
        print(f"you have reached majority for {yearSM} years :)")
else:
        print("Cool, a young one :)")
        days = age * 365
        print(f"Only a bit more than {days} days since your were born")
```

← What will happen if I remove the typecasting here?

Note on Syntax

Do not forget the colon! :

```
1 while condition:
2     instruction
```



Iterations: The while loop

```
x = 12
y = 3
res = "dummy"
while res != x*y:
    if res == "dummy":
        print(f"What is {x} x {y}?")
    else:
        print("you're wrong!!! Try again:")
    res = int(input(f"{x}x{y}="))
print(f"well done! It is {res}!")
```

Iterations: The while loop

```
x = 12
y = 3
res = "dummy"
while res != x*y:
  if res == "dummy":
    print(f"What is {x} x {y}?")
  else:
    print("you're wrong!!! Try again:")
  res = int(input(f"{x}x{y}="))
print(f"well done! It is {res}!")
```

condition

Iterations: The while loop

```
x = 12
y = 3
res = "dummy"
while res != x*y:
  if res == "dummy":
    print(f"What is {x} x {y}?")
  else:
    print("you're wrong!!! Try again:")
  res = int(input(f"{x}x{y}="))
print(f"well done! It is {res}!")
```

condition

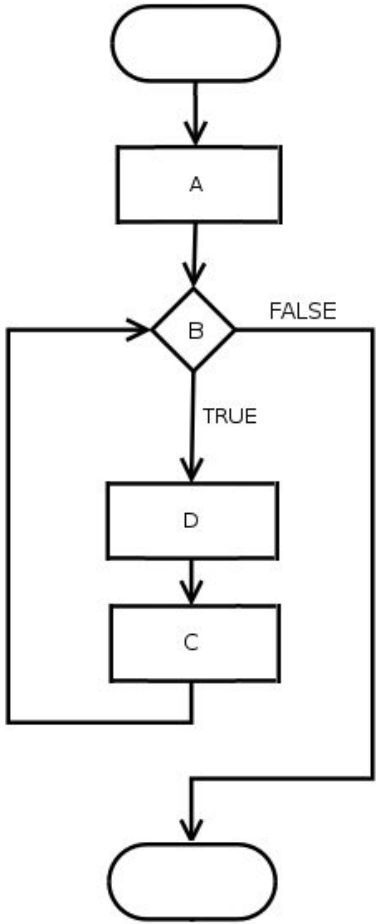
code that will be repeated for as long as the condition is true

Note on Syntax

Do not forget the colon! :

```
1  for element in mylist:
2      instruction
```

for(A;B;C)
  D;



Iterations: the for loop

```
import math
y = int(input("number: "))
for x in range(1, 1000):
 if int(math.log(x, y)) == math.log(x,y):
   # condition above could be done with isInteger()
   print(f"{x} is a power of {y}")
```

Iterations: the for loop

```
import math
y = int(input("number: "))
for x in range(1, 1000):
 if int(math.log(x, y)) == math.log(x,y):
   # condition above could be done with isInteger()
   print(f"{x} is a power of {y}")
```

All numbers between 0 and 999, incrementing by one

Iterations: the for loop

```
import math
y = int(input("number: "))
for x in range(1, 1000):
 if int(math.log(x, y)) == math.log(x,y):
  # condition above could be done with isInteger()
  print(f"{x} is a power of {y}")
```

Repeat for x taking values in

All numbers between 0 and 999, incrementing by one

Iterations: the for loop

```
import math
y = int(input("number: "))
for x in range(1, 1000):
 if int(math.log(x, y)) == math.log(x,y):
  # condition above could be done with isInteger()
  print(f"{x} is a power of {y}")
```

Repeat for x taking values in
All numbers between 0 and 999, incrementing by one

Code being repeated with x=0, x=1, x=2, …, x=999

Iterations: the for loop

```
for x in range(0,100):
 print(f"x is {x}")
```

is exactly the same:

```
x = 0
while x < 100:
 print(f"x is {x}")
 x = x + 1
```

Iterations: the for loop

- Can be used on lists

```
1  Students = ["Mina", "Anna", "Max", "Aurore"]
2  for student in students:
3          print(student)
```

On range*(s,e,i)*

The **for** loop iterate giving a variable new values in a list or sequence.

```
range(start, stop, increment)
```

Is used here to generates a sequence of numbers, starting with `start` and finishing just before `stop`.

range(10), numbers from 0 to 9, incremented by 1 (i.e. start is 0 by default)

range(20, 30), numbers from 20 to 29 incremented by 1 (i.e. increment is 1 by default)

range(38, 11, -2), numbers from 38 to 12, decreasing by 2

Iterations: break and continue

Break - stop the loop immediately (whether it is a for or a while)

```
# find multiples of 128 but stop of encountering a power of 12 greater than 200
for x in range(129, 1000000):
 if x % 128 == 0:
   print(f"{x} is a multiple of 128")
 if math.log(x, 12).is_integer() and x > 200:
   print(f"{x} is a power of 12")
   break
```

Iterations: break and continue

Break - stop the loop immediately (whether it is a for or a while)

```
# find multiples of 128 but stop of encountering a power of 12 greater than 200
for x in range(129, 1000000):
 if x % 128 == 0:
   print(f"{x} is a multiple of 128")
 if math.log(x, 12).is_integer() and x > 200:
   print(f"{x} is a power of 12")
   break
```

Iterations: break and continue

Continue - interrupts the current iteration and goes back at the beginning of the loop (with a new value in the case of a for loop)

```
# rebuild a string without space
newstring = ""
for c in "this is a beautiful string":
 if c == " ":
   continue
 newstring = newstring+c
print(newstring)
```

Iterations: break and continue

Continue - interrupts the current iteration and goes back at the beginning of the loop (with a new value in the case of a for loop)

```
# rebuild a string without space
newstring = ""
for c in "this is a beautiful string":
 if c == " ":
  continue
 newstring = newstring+c
print(newstring)
```

Note

```
newstring = ""
for c in "this is a beautiful string":
 if c == " ":
   continue
 newstring = newstring+c
print(newstring)
```

Is exactly the same as

```
newstring = ""
for c in "this is a beautiful string":
  if c != " ":
    newstring = newstring+c
print(newstring)
```

Note on Syntax

Do not forget the colon! :

```
1 if condition:
2     instruction
3 elif condition:
4     instruction
5 else:
6     instruction
```

```
1 while condition:
2     instruction
```

```
1 for element in mylist:
2     instruction
```

To be seen in labs

Some more interesting algorithms using conditionals and loops and possibly strings