UNIVERSITÉ DE LORRAINE

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

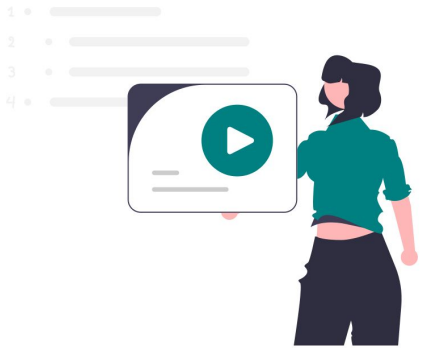
Programming

5- Tuples, Recursion, Files

These slides will be available on Arche

By Gaël Guibon, inspired by Mathieu d'Aquin's course

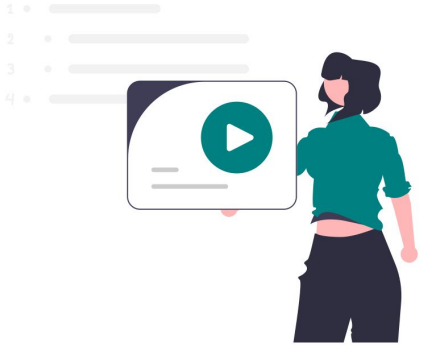
Reminder on Functions



Functions

Reusable, “callable”, parameterisable pieces of code enabling to structure code through **decomposition** and **abstraction**.

```
def searchIndex(mylist,tosearch,searchfrom=0):
 """
 Function to search for the index of an
 element (tosearch) in a list (mylist)
 """
 for i in range(searchfrom, len(mylist)):
   if mylist[i] == tosearch:
     return i
```

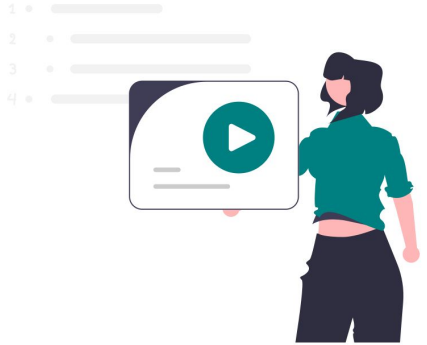


Functions

Reusable, “callable”, parameterisable pieces of code enabling to structure code through **decomposition** and **abstraction**.

name of the function

```
def searchIndex(mylist,tosearch,searchfrom=0):
  """
  Function to search for the index of an
  element (tosearch) in a list (mylist)
  """
  for i in range(searchfrom, len(mylist)):
    if mylist[i] == tosearch:
      return i
```



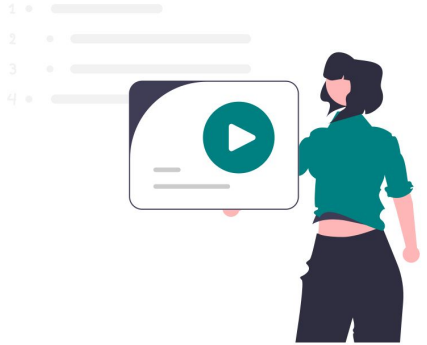
Functions

Reusable, “callable”, parameterisable pieces of code enabling to structure code through **decomposition** and **abstraction**.

name of the function

fixed arguments

```
def searchIndex(mylist,tosearch,searchfrom=0):
 """
 Function to search for the index of an
 element (tosearch) in a list (mylist)
 """
 for i in range(searchfrom, len(mylist)):
   if mylist[i] == tosearch:
     return i
```



Functions

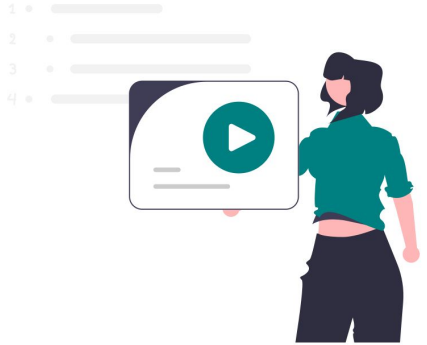
Reusable, “callable”, parameterisable pieces of code enabling to structure code through **decomposition** and **abstraction**.

name of the function

default (optional) arguments

fixed arguments

```
def searchIndex(mylist,tosearch,searchfrom=0):
 """
 Function to search for the index of an
 element (tosearch) in a list (mylist)
 """
 for i in range(searchfrom, len(mylist)):
   if mylist[i] == tosearch:
     return i
```



Functions

Reusable, “callable”, parameterisable pieces of code enabling to structure code through **decomposition** and **abstraction**.

name of the function

default (optional) arguments

```
def searchIndex(mylist,tosearch,searchfrom=0):
 """
 Function to search for the index of an
 element (tosearch) in a list (mylist)
 """
 for i in range(searchfrom, len(mylist)):
   if mylist[i] == tosearch:
     return i
```

fixed arguments

documentation string for the function

Functions

Reusable, “callable”, parameterisable pieces of code enabling to structure code through **decomposition** and **abstraction**.

name of the function

default (optional) arguments

```
def searchIndex(mylist,tosearch,searchfrom=0):
    """
    Function to search for the index of an
    element (tosearch) in a list (mylist)
    """
    for i in range(searchfrom, len(mylist)):
        if mylist[i] == tosearch:
            return i
```

fixed arguments

documentation string for the function

returned value for the function

Functions

Reusable, “callable”, parameterisable pieces of code enabling to structure code through **decomposition** and **abstraction**.

name of the function

default (optional) arguments

```
def searchIndex(mylist,tosearch,searchfrom=0):
    """
    Function to search for the index of an
    element (tosearch) in a list (mylist)
    """
    for i in range(searchfrom, len(mylist)):
        if mylist[i] == tosearch:
            return i
```

fixed arguments

documentation string for the function

returned value for the function

if we reach the end of the function without return, `None` will be returned

Functions

Reusable, “callable”, parameterisable piece
structure code through **decomposition** an

```
def searchIndex(mylist,tosearch,searchfrom=0):
 """
 Function to search for the index of an
 element (tosearch) in a list (mylist)
 """
 for i in range(searchfrom, len(mylist)):
   if mylist[i] == tosearch:
     return i
```

```
l = ["a","b","c","d","a","b","c"]
i = searchIndex(l, "c")
print(i)
print(searchIndex(l, "b"))
print(searchIndex(l, "b",2))
print(l[searchIndex(l, "b",2)+1])
print(searchIndex(l, "b",
              searchIndex(l, "b")+1))
print(searchIndex(l, "z"))
```

```
2
1
5
c
5
None
```

Tuples

Tuples

A tuple is an **immutable** sequence of values, which can be of any type (and even mixed). A tuple can also be of any size.

```
t = ("Anna", 12, True, 3.5, 1)

type(t) # type of t
len(t) # size of t
t[3] # element at index 3
t[1:4] # between 1 and 3
t[-1] # last element
t[::-1] # t in reverse
3.5 in t # check inclusion
"joee" not in t # check inclusion
```

Tuples

A tuple is an **immutable** sequence of values, which can be of any type (and even mixed). A tuple can also be of any size.

```
t = ("Anna", 12, True, 3.5, 1)

type(t) # type of t >>> tuple
len(t) # size of t >>> 5
t[3] # element at index 3 >>> 3.5
t[1:4] # between 1 and 3 >>> (12, True, 3.5)
t[-1] # last element >>> 1
t[::-1] # t in reverse >>> (1, 3.5, True, 12, "Anna")
3.5 in t # check inclusion >>> True
"joee" not in t # check inclusion >>> True
```

Tuples

I.e. you cannot modify it

A tuple is an **immutable** sequence of values, which can be of any type (and even mixed). A tuple can also be of any size.

```
t = ("Anna", 12, True, 3.5, 1)

type(t) # type of t >>> tuple
len(t) # size of t >>> 5
t[3] # element at index 3 >>> 3.5
t[1:4] # between 1 and 3 >>> (12, True, 3.5)
t[-1] # last element >>> 1
t[::-1] # t in reverse >>> (1, 3.5, True, 12, "Anna")
3.5 in t # check inclusion >>> True
"joee" not in t # check inclusion >>> True
```

What you cannot do with tuples

```
t = ("Anna", 12, True, 3.5, 1)
t[2] = 13
```

```
_____________________________________
      1 t = ("Hanna", 12, True, 3.5, 1)
----> 2 t[2] = 13

TypeError: 'tuple' object does not support item
assignment
```

Tuples and Functions

Multiple elements returned lead to a **tuple**

(val1**,** val2**)**

```
charmander = {"type": "🔥", "hp": 100}
squirtle = {"type": "🌊", "hp": 100}

def attack(sender, receiver):
    damage = 0
    if sender["type"] == "🔥" and receiver["type"] == "🌊":
        damage = 10
    elif sender["type"] == "🌊" and receiver["type"] == "🔥":
        damage = 100
    receiver['hp'] -= damage
    return sender, receiver

print("charmander attacks squirtle")
print(attack(charmander, squirtle))
```

Tuples and Functions

Multiple elements returned lead to a **tuple**

(val1**,** val2**)**

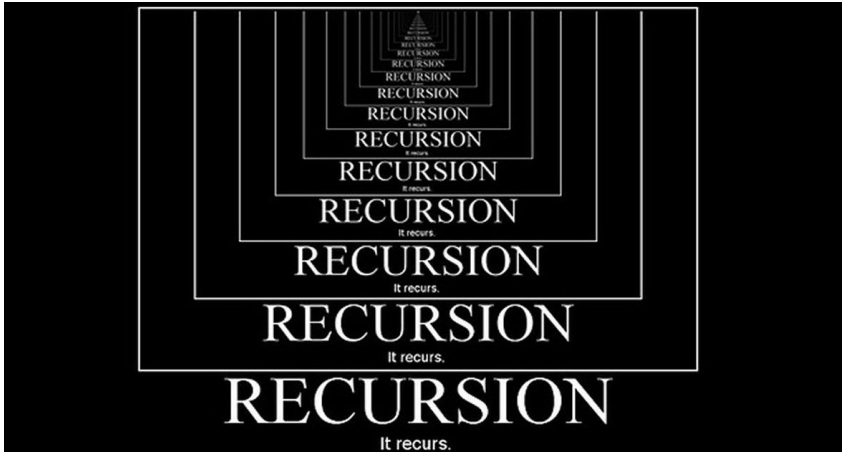
```
charmander = {"type": "🔥", "hp": 100}
squirtle = {"type": "🌊", "hp": 100}

def attack(sender, receiver):
    damage = 0
    if sender["type"] == "🔥" and receiver["type"] == "🌊":
        damage = 10
    elif sender["type"] == "🌊" and receiver["type"] == "🔥":
        damage = 100
    receiver['hp'] -= damage
    return sender, receiver

print("charmander attacks squirtle")
print(attack(charmander, squirtle))
```

```
charmander attacks squirtle
({'type': '🔥', 'hp': 100}, {'type': '🌊', 'hp': 90})
```

Recursion



What is recursion?

Technically: A programming technique where a function calls itself 🤯

Algorithmically: A way to design solutions to problems by divide-and-conquer: **reduce a problem to simpler versions of the same problem**

Also: Often a more elegant, easier to implement (but harder to debug) alternative to iterations (loops).

Example: Multiplication

An **iterative** implementation of v1xv2 : add v1 v2 times

```
def iterMul(v1,v2):
    """iterative multiplication"""
    res = 0
    for i in range(v2):
    res += v1
    return res

print(iterMul(5,4)) >>> 20
print(iterMul(11,8)) >>> 88
print(iterMul(123,45)) >>> 5535
```

Example: Multiplication

A **recursive** implementation of v1xv2 : recursively do v1+v1x(v2-1)

```
def recMul(v1,v2):
    """recursive multiplication"""
    if v2 == 1: return v1
    return v1+recMul(v1,v2-1)

print(recMul(5,4)) >>> 20
print(recMul(11,8)) >>> 88
print(recMul(123,45)) >>> 5535
```

Example: Multiplication

A **recursive** implementation of v1xv2 : recursively do v1+v1x(v2-1)

```
def recMul(v1,v2):
    """recursive multiplication"""
    if v2 == 1: return v1
    return v1+recMul(v1,v2-1)


print(recMul(5,4)) >>> 20
print(recMul(11,8)) >>> 88
print(recMul(123,45)) >>> 5535
```

Building a recursive solution

2 elements :

- A (sometimes several) **base case**(s) or stop condition(s), i.e. cases where the solution is known and does not need further recursion
- A (sometimes several) **recursion step**(s), i.e. way(s) to reduce the problem to a smaller problem when not in the base case

Building a recursive solution

Multiplication:

- **Base case**: multiplication by 1
- **Recursion step**: multiply by v2-1

Validating a recursive algorithm

3 steps:

- Prove that the best case is correct
- Prove that if the result of a recursive call is correct, then the result of using it is correct
- Prove that successive recursive steps will converge to the base case

Validating a recursive algorithm

3 steps:

- Prove that the best case is correct
- Prove that if the result of a recursive call is correct, then the result of using it is correct
- **Prove that successive recursive steps will converge to the base case**

Validating a recursive algorithm

3 steps:

- Prove that the best case is correct
- Prove that if the result of a recursive call is correct, then the result of using it is correct
- **Prove that successive recursive steps will converge to the base case**

Important to avoid infinite recursion, which is bad...

Validating a recursive algorithm

Validating a recursive algorithm

Multiplication:

- v1x1 = 1 is correct
- if v3 is v1x(v2-1), then v1+v3 is v1xv2
- For any number (v2), successively removing 1 will eventually reach 1

Validating a recursive algorithm

Multiplication:

- v1x1 = 1 is correct
- if v3 is v1x(v2-1), then v1+v3 is v1xv2
- For any number (v2), successively removing 1 will eventually reach 1

Hummm… really?

Validating a recursive algorithm

Multiplication:

- v1x1 = 1 is correct
- if v3 is v1x(v2-1), then v1+v3 is v1xv2
- For any **positive non 0** number (v2), successively removing 1 will eventually reach 1

Validating a recursive algorithm

Multiplication:

- v1x1 = 1 is correct
- if v3 is v1x(v2-1), then v1+v3 is v1xv2
- For any **positive non 0** number (v2), successively removing 1 will eventually reach 1

```
recMul(3,-4)

__________________
----> 4    return v1+recMul(v1,v2-1)
RecursionError: maximum recursion depth exceeded in comparison
```

Recursive “search in list” when the list is sorted

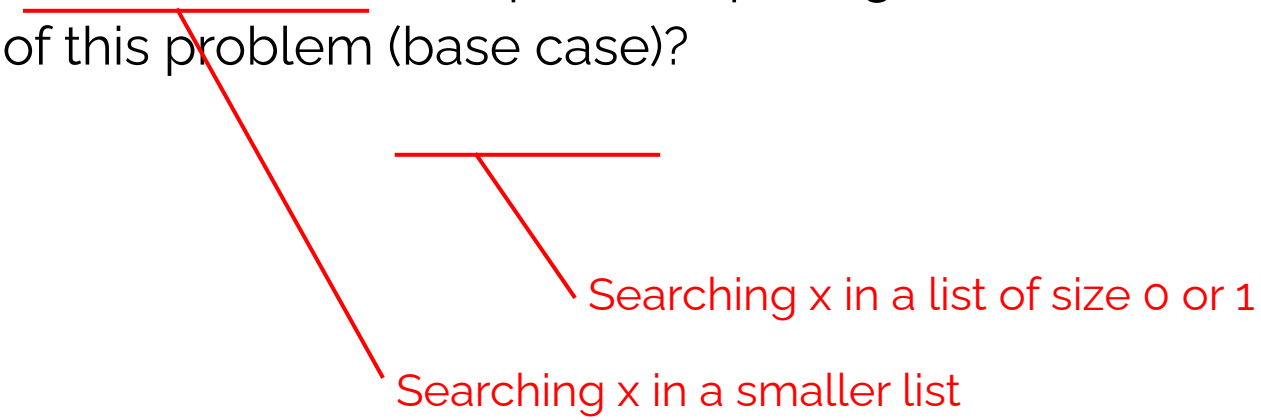
Given a list `l` sorted in ascending order, check if an element `x` is in it

- How can we reduce this problem into a simpler sub-problem (recursion step)?
- When should we stop decomposing? What is the simplest version of this problem (base case)?

Recursive “search in list” when the list is sorted

Given a list `l` sorted in ascending order, check if an element `x` is in it

- How can we reduce this problem into a simpler sub-problem (recursion step)?
- When should we stop decomposing? What is the simplest version of this problem (base case)?



Files

Files

Files are the primary and most common way to **import data into your program**.

Here we will mostly see text files but the same principles apply to other file formats.

The code for this lecture is to be **run locally with the python interpreter** as files, modules and packages are easier to understand this way than in notebooks.

Example

file1.txt

```
This is what my file contains:

First there are the two first lines,
including this one.


Then, there is an empty line, and
this line.

Then the last line comes in, just to
say goodbye.
```

```python
f = open("file1.txt")
lines = f.readlines()
print(lines)
for i,l in enumerate(lines):
  print(f"Line {i}: {l}")
f.close()
```

Example

```
f = open("file1.txt")
lines = f.readlines()
print(lines)
for i,l in enumerate(lines):
  print(f"Line {i}: {l}")
f.close()
```

```
['This is what my file contains:\n', 'First there are
the two first lines, including this one.\n', '\n',
'Then, there is an empty line, and this line.\n', 'Then
the last line comes in, just to say goodbye.\n']

Line 0: This is what my file contains:

Line 1: First there are the two first lines, including
this one.

Line 2:

Line 3: Then, there is an empty line, and this line.

Line 4: Then the last line comes in, just to say
goodbye.
```

Example

```
f = open("file2.txt", "w")
f.write("I'm writing in a file! \n")
f.write("Doing it again! \n")
f.write("Getting a bit boring
now.\n")
f.write("...")
f.write("done now...")
f.close()
```

File2.txt:

```
I'm writing in a file!
Doing it again!
Getting a bit boring now.
...done now...
```

Opening a file

```
f = open("file1.txt")
f = open("file2.txt", "w")
```

By default, a file is open for reading only. The second parameter concerns the mode of the file, which can be read, write, read/write and others.

“w” means write and overwrite if already exists.

Existing modes

r	Opens a file for **reading**. (default)
w	Opens a file for **writing.** Creates a new file if it does not exist or truncates the file if it exists.
x	Opens a file for **exclusive creation**. If the file already exists, the operation fails.
a	Opens a file for **appending** at the end of the file without truncating it. Creates a new file if it does not exist.
t	Opens in **text mode**. (default)
b	Opens in **binary mode**.
+	Opens a file for **updating** (reading and writing)

Existing modes: the ones you will use often

r	Opens a file for **reading**. (default)
w	Opens a file for **writing.** Creates a new file if it does not exist or truncates the file if it exists.
x	Opens a file for **exclusive creation**. If the file already exists, the operation fails.
a	Opens a file for **appending** at the end of the file without truncating it. Creates a new file if it does not exist.
t	Opens in **text mode**. (default)
b	Opens in **binary mode**.
+	Opens a file for **updating** (reading and writing)

Combining modes

Some modes are in different categories and are meant to be combined. For example,

```
f = open("image.jpg", "r+b")
```

Opens a file for reading and writing, without overwriting it if it exists (it must exist), as a binary file.

More on modes

	read	write	create	overwrite	start beg.	start end.
r	X				X	
r+	X	X			X	
w		X	X	X	X	
w+	X	X	X	X	X	
a		X	X			X
a+	X	X	X			X

Encodings

Another parameter of open is the encoding to use to decode the file.

With a (unicode encoded file) like:

File3.txt

```
A french sentence: "j'ai mangé du fromage"

Here is the cheese in question: 🧀
```

Encodings

By default, python3 will open it with the `utf-8` (unicode) encoding.

```
f = open("file3.txt", "r")
for l in f.readlines():
   print(l)
f.close()
```

```
_____________________________
0 : A french sentence: "j'ai mangé du fromage"

1 : Here is the cheese in question: 🧀
```

Encodings

By default, python3 will open it with the `utf-8` (unicode) encoding.

```
f = open("file3.txt", encoding="utf-8")
for l in f.readlines():
   print(l)
f.close()
```

```
_____________________________
0 : A french sentence: "j'ai mangé du fromage"

1 : Here is the cheese in question: 🧀
```

Encodings

But we can use others, e.g. `latin1`

```
f = open("file3.txt", encoding="latin1")
for l in f.readlines():
   print(l)
f.close()
```

```
_____________________________
0 : A french sentence: "j'ai mangÃ© du fromage"

1 : Here is the cheese in question: ð
```

Encodings

But we can use others, e.g. `ascii` (default in python2)

```
f = open("file3.txt", encoding="ascii")
for l in f.readlines():
    print(l)
f.close()
```

```
_____________________
Traceback (most recent call last):
  File "/home/mdaquin/teaching/prog/S7/file3.py", line 15, in <module>
    for i,l in enumerate(f.readlines()):
  File "/home/mdaquin/miniconda3/lib/python3.9/encodings/ascii.py", line 26,
in decode
    return codecs.ascii_decode(input, self.errors)[0]
UnicodeDecodeError: 'ascii' codec can't decode byte 0xc3 in position 29:
ordinal not in range(128)
```

Reading in a file

`read()`	Reads from the current position, to the end of the file. Returns a string if the file is open in text mode and a byte array in binary mode.
`read(n)`	Same as above but only reads n characters/bytes. Returns empty string/byte array if reached the end of the file.
`readline()`	Read from the current position to the next “newline” character.
`readlines()`	Returns an array with all the lines from the current position to the end of the file.
`tell()`	Returns the current position in the file.
`seek(p)`	Go to position p in the file.

Example

```
f = open("file3.txt")
f.seek(10)
print(f.read(5))
print(f.tell())
print(f.readline())
f.seek(10)
print(f.readlines())
f.close()
```

```
enten
15
ce: "j'ai mangé du fromage"

['entence: "j\'ai mangé du
fromage"\n', 'Here is the cheese
in question: 🧀 \n']
```

Writing in a file

`write(s)`	Write the string or value or byte array s into the file at the end of the file. If the file is open with w or w+, it will be empty.

Example

```
f= open("file4.txt", "a+")
f.write("Alicia")
f.seek(0)
print(f.read())
f.seek(0)
f.write("Inès")
f.close()
```

file4.txt at the start:

```
Hello,
nice to meet you.
```

Output:

```
Hello,
nice to meet you.
Alicia
```

File4.txt at the end:

```
Hello,
nice to meet you.
Inès
```

Closing a file

It is not a good idea to leave a file open:

- It keeps resources open that don't need to be
- It locks the file for other programmes

`close()`	Closes the file. Read and write operations stop working on the file once this is closed.

The `with` clause

Creates a code block in which the file is open, and then closes it afterwards.

```
with open("file3.txt") as f:
    line1 = f.readline()
    line2 = f.readline()


print(f.close())
print("line 1:", line1)
print("line 2:", line2)
```

```
True
line 1: A french sentence: "j'ai
mangé du fromage"

line 2: Here is the cheese in
question: 🧀
```

Example: csv file

A csv file is a file with tabular data where each line is a row and columns are separated by commas, e.g. the file **mk_bodies.csv**

```
Vehicle,Speed,Acceleration,Weight,Handling,Traction,Mini Turbo
Standard Kart,0,0,0,0,0,0
Pipe Frame,-0.5,0.5,-0.25,0.5,0.25,0.5
Mach 8,0,-0.25,0.25,-0.25,0.25,0
Cat Cruiser,-0.25,0.25,0,0.25,0,0.25
Steel Driver,0.25,-0.75,0.5,-0.5,0,-0.5
Circuit Special,0.5,-0.75,0.25,-0.5,-0.5,-0.75
Tri-Speeder,0.25,-0.75,0.5,-0.5,0,-0.5
Badwagon,0.5,-1,0.5,-0.75,0.5,-1
Prancer,0.25,-0.5,-0.25,0,-0.25,-0.25
Biddybuggy,-0.75,0.75,-0.5,0.5,0.5,0.25
...
```

Reading a csv file

```
data = []
with open("mk_bodies.csv") as f:
    line = f.readline()
    while line:
        data.append(line.strip().split(","))
        line = f.readline()

print(data)
```

[['Vehicle', 'Speed', 'Acceleration', 'Weight', 'Handling', 'Traction', 'Mini Turbo'], ['Standard Kart', '0', '0', '0', '0', '0', '0'], ['Pipe Frame', '-0.5', '0.5', '-0.25', '0.5', '0.25', '0.5'], ['Mach 8', '0', '-0.25', '0.25', '-0.25', '0.25', '0'], ['Cat Cruiser', '-0.25', '0.25', '0', '0.25', '0', '0.25'], ['Steel Driver', '0.25', '-0.75', '0.5', '-0.5', '0', '-0.5'], ['Circuit Special', '0.5', '-0.75', '0.25', '-0.5', '-0.5', '-0.75'], ['Tri-Speeder', '0.25', '-0.75', '0.5', '-0.5', '0', '-0.5'], ['Badwagon', '0.5', '-1', '0.5', '-0.75', '0.5', '-1'], ['Prancer', '0.25', '-0.5', '-0.25', '0', '-0.25', '-0.25'], ['Biddybuggy', '-0.75', '0.75', '-0.5', '0.5', '0.5', '0.25'], ['Landship', '-0.5', '0.5', '-0.5', '-0.5', '0.75', '0.5'], ['Sneeker', '0.25', '-0.5', '0', '0', '-0.75', '-0.25'], ['Sports Coupe', '0', '-0.25', '0.25', '-0.25', '0.25', '0'], ['Gold Standard', '0.25', '-0.5', '0', '0', '-0.75', '-0.25'], ['Mercedes GLA', '0.5', '-1', '0.5', '-0.75', '0.5', '-1'], ['Mercedes Silver Arrow', '-0.25', '0.25', '-0.25', '0.25', '0.5', '0.25'], ['Mercedes 300 SL Roadster', '0', '0', '0', '0', '0', '0'], ['Blue Falcon', '0.25', '-0.25', '-0.5', '-0.25', '0.5', '0'], ['Tanooki Kart', '-0.25', '-0.5', '0.25', '0.25', '0', '1'], ['B Dasher', '0.5', '-0.75', '0.25', '-0.5', '-0.25', '-0.5'], ['Streetle', '-0.5', '0.5', '-0.5', '-0.5', '-0.25', '0.75'], ['P-Wing', '0.5', '-0.75', '0.25', '-0.5', '-0.25', '-0.5'], ['Koopa Clown', '-0.25', '-0.5', '0.25', '0.25', '0', '1'], ['Standard Bike', '-0.25', '0.25', '-0.25', '0.25', '0.5', '0.25'], ['Comet', '-0.25', '0.25', '0', '0.25', '0', '0.25'], ['Sport Bike', '0.25', '-0.5', '-0.25', '0'

How to Store Dictionaries?

Using the `json` module to write a dictionary into a file

```
import json


f = open("mydata.json", "w")
dico = {"firstname": "tiankai", "lastname": "luo"}
json.dump(dico,f)
f.close()
```

How to Store Dictionaries?

Using the `json` module to write a dictionary into a file

```
import json


f = open("mydata.json", "w")
dico = {"firstname": "tiankai", "lastname": "luo"}
json.dump(dico,f)
f.close()
```

Now I have a file with this dictionary as a content.

How to Read Dictionaries?

Using the `json` module to read data directly as dictionary

```
f = open("mydata.json", "r")

data = json.load(f)

f.close()
```

How to Read Dictionaries?

Using the `json` module to read data directly as dictionary

```
f = open("mydata.json", "r")

data = json.load(f) # {"firstname": "tiankai", "lastname": "luo"}

f.close()
```

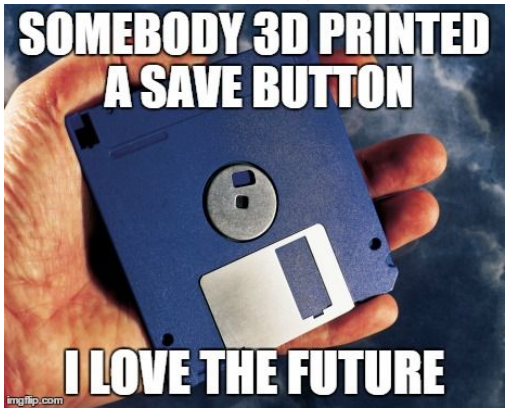
To be seen in lab

Browse some external data 💾

Make your game code distinct from your game content !

To be seen in lab

Browse some external data 💾



Make your game code distinct from your game content !