

# Python Basics I

13.11.19 / Us



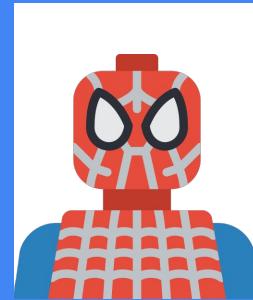
# Team



Hamid



Xavi



Vincent

# What is the plan...



## Basics I

06.11.19  
AND  
13.11.19



## Basics II

20.11.19



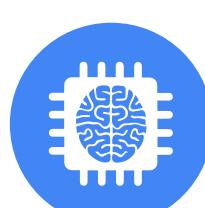
## NumPy

27.11.19



## Pandas

04.12.19



## Machine Learning

11.12.19

# GitLab:

# <https://tinyurl.com/y5y7oj6n>



## Basics I

06.11.19  
AND  
13.11.19



## Basics II

20.11.19



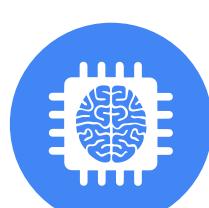
## NumPy

27.11.19



## Pandas

04.12.19



## Machine Learning

11.12.19

# What is Python?!

- General Purpose Language
- Object oriented (e.g. inheritance)
- Open Source and Multi-Platform
- Libraries, Third Party Utilities, Built-in Types and Tools
- \*Monty Python



# Installing Python



- Python is default pre-installed in most of the UNIX based systems → Linux & MacOs
- **Download Link:**  
<https://www.python.org/downloads/>



- **Download Link:**  
<https://www.python.org/downloads/>
  - Latest version is Python 3.7.3
- Edit Environment Variable to run python scripts from the terminal // **or accept in Download process!**

# We are gonna do it a bit differently...



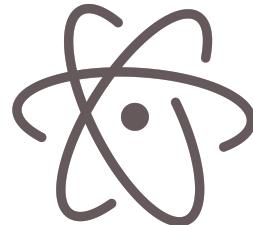
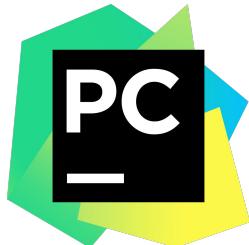
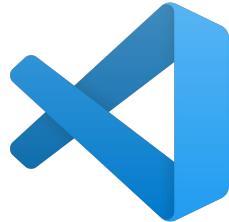
- **Why Anaconda!**

- Easy python install
- 720 open source packages (e.g. NumPy, Matplotlib, etc.)
- Conda package manager → easy use of virtual environment

- **Why Jupyter Notebook**

- Cool environment where you can run your code, look at the results & easily modify
- Easy sharing & virtual environment
- Nice take-away!

# But what about other stuff?



- **Code Editors with Python support**
  - Text editor with syntax highlighting
  - Visual Studio Code, Atom Editor
  
- **Python-Specific IDEs**
  - Program dedicated to software development  
(code editor + execution/debugging tools)
  - PyCharm, Spyder

# Syntax

- **Indentation:**
  - Structures the blocks of your code - number of spaces is variable, but code within the same block must be indented the same amount - best practice: 4 spaces
- **The header line:**
  - For compound statements such as: while, if, def, etc. must be terminated with a colon (:)
- **Printing on the screen**
  - `print("Hello World")`
- **Reading the keyboard**
  - `Fav_Number = input("Tell me your number? ")`
- **Comments:**
  - Single line → `# This is just a random comment`
  - Multiple lines → `'''Oh my good, this is awesome'''`
- **Python files**
  - End with `.py`

# Variables

- **Dynamically typed:**
  - You don't have to declare the type of your variable
- **Automatically change:**
  - You can easily change the data type of your variable by just assigning it newly
- **Single value for all:**
  - In Python you can assign a single value to multiple variables
- **Multiple objects to multiple variables:**
  - Further, you can assign multiple objects to multiple variables
- **Naming convention:**
  - Lowercase single letter, separate words with underscores (\_) to improve readability
  - For Class → CamelCase

# Data Types

# Numbers

- Immutable in python  
→ cannot change their values
- 3 built-in data types:
  - Integers
  - Floating point numbers
  - Complex numbers

Functions	Explanation
int(x)	Convert x in integer
float(x)	Convert x in float
abs(x)	Absolute value of x
exp(x)	Exponential of x
log(x)	Natural logarithm of x
pow(x, y)	Value of $x^{**}y$
sqrt(x)	Square root of x

# Math Operations in Python

<b>+ Addition</b>	Adds values on either side of the operator.	$a + b = 30$
<b>- Subtraction</b>	Subtracts right hand operand from left hand operand.	$a - b = -10$
<b>* Multiplication</b>	Multiplies values on either side of the operator	$a * b = 200$
<b>/ Division</b>	Divides left hand operand by right hand operand	$b / a = 2$
<b>% Modulus</b>	Divides left hand operand by right hand operand and returns remainder	$b \% a = 0$
<b>** Exponent</b>	Performs exponential calculation on operators	$a^{**}b = 10 \text{ to the power } 20$
<b>//</b>	Floor Division	$9 // 2 = 4$

# Strings

- Immutable in python
- Update a string by assign a variable to another string
- No <>char>> type
- Single, double or triple quotes for strings
- Index starts from 0 and -1 from the end along

Functions & Methods	Explanation
<code>str(x)</code>	Convert x in string
<code>len(string)</code>	Total length of string
<code>string.count("am")</code>	Counts "am" in string
<code>string.lower()</code>	Converts string to lowercase
<code>string.upper()</code>	Converts string to uppercase
<code>string.replace(old, new)</code>	Replaces all occurrences in string with new string
<code>string.split(separator, max)</code>	Splits string with defined separator and gives list
<code>string.strip()</code>	Removes all leading and trailing whitespaces in string

# String Operators I

## String Operators

### Concatenation

Adds the value on both sides of the operator

```
In [2]: string_1 = 'Hello'  
        string_2 = 'Python Master!'  
        string_sum = string_1 + string_2  
        print(string_sum)
```

```
HelloPython Master!
```

### Repetition

```
In [3]: string_1 = 'movies'  
        string_2 = string_1*2  
        print(string_2)
```

```
moviesmovies
```

# String Operators II

## List Slicing

Slices through the string like a list. Range slice gives you the characters from a given range, where just the index gives you the referring character

```
In [10]: string_1 = "Apfelkuchen"
index_1 = string_1[0]
index_2 = string_1[-1]
range_1 = string_1[1:4]
range_2 = string_1[:5]
print('Hier ist der Index 1: ' + index_1)
print('Hier ist der Index 2: ' + index_2)
print('Hier ist die Range 1: ' + range_1)
print('Hier ist die Range 2: ' + range_2)
```

```
Hier ist der Index 1: A
Hier ist der Index 2: n
Hier ist die Range 1: pfe
Hier ist die Range 2: Apfel
```

## Membership with in

Checks if the character exists in the given and return True, if it is the case...

```
In [9]: string_1 = "Allerlei Sachen"
condition_1 = "All" in string_1
print(condition_1)
```

```
True
```

# String Formatting in Python

## String Formatting

There are two ways in order to format your strings. The difference comes with the updates version of python (Python 3), where f-strings are introduced for the first time. However, with the old version, you cannot fail.

## Old School String Formatting

```
In [1]: num = 10
string_1 = "I have {} favourite movies".format(num)
print(string_1)
```

```
I have 10 favourite movies
```

## F-String Python 3

```
In [2]: num = 10
string_1 = 'movies'
string_2 = f"I have {num} favourite {string_1}"
print(string_2)
```

```
I have 10 favourite movies
```

# Challenge 1



# Lists

- Mutable in python
- Ordered group of elements - don't have to be same data type
- Most string operations work for lists
- Can have sublists, and sublists of sublists

Functions & Methods	Explanation
<code>len(list)</code>	Total length of list
<code>cmp(list1,list2)</code>	Compares elements of both lists
<code>max(list)</code>	Return item with highest value
<code>min(list)</code>	Return item with lowest value
<code>list.append()</code>	Appends element to list
<code>list.insert(index, element)</code>	Inserts element to list according to index
<code>list.remove(element)</code>	Removes element from list
<code>list.sort()</code>	Sorts list in place
<code>list.reverse()</code>	Reverses elements in list in place

# List Comprehension

## List Comprehension

Each list comprehension contains an **expression** and then a **for** clause

```
In [11]: list_1 = [1,2,3,4,5]
          list_2 = [x+1 for x in list_1]
          print(list_2)
```

```
[2, 3, 4, 5, 6]
```

```
In [12]: list_1 = [1,2,3,4,5]
          list_2 = [x+1 for x in list_1]
          list_3 = [z*2 for z in [x+1 for x in list_1]]
          print(list_3)
```

```
[4, 6, 8, 10, 12]
```

```
In [ ]:
```

# List Operations I

## List Operations

Just a quick summary of most common list operations

### Append, Insert & Remove

!Wow, the same list is showed twice, why?!

```
In [3]: list_1 = [90,12,3,[ "Katze",2], "Hund"]
list_2 = list_1
list_1.append(99)
list_2.insert(2,'Element')

print(list_1)
print(list_2)
```

```
[90, 12, 'Element', 3, ['Katze', 2], 'Hund', 99]
[90, 12, 'Element', 3, ['Katze', 2], 'Hund', 99]
```

Now, we will make some small changes!

```
In [4]: list_1 = [90,12,3,[ "Katze",2], "Hund"]
list_2 = list_1.copy()
list_1.append(99)
list_2.insert(2,'Element')

print(list_1)
print(list_2)

[90, 12, 3, ['Katze', 2], 'Hund', 99]
[90, 12, 'Element', 3, ['Katze', 2], 'Hund']
```

```
In [5]: list_1 = [90,12,3,[ "Katze",2], "Hund"]
list_1.remove(90)
print(list_1)

[12, 3, ['Katze', 2], 'Hund']
```

# List Operations II

## List Operations

Just a quick summary of most common list operations

### Sort & Reverse

first let us look at sort

```
In [18]: list_1 = [90,12,20,23]
list_2 = ['Manfred', 'Stefan', 'Lisa', 'Hanna',]

list_1.sort()
list_2.sort()
print(list_1) #from bottom to top
print(list_2)

list_1.sort(reverse = True) #from top to bottom
print(list_1)

list_2 = [90,12,20,23]

[12, 20, 23, 90]
['Hanna', 'Lisa', 'Manfred', 'Stefan']
[90, 23, 20, 12]
```

now reverse

```
In [19]: list_1 = [90,12,20,23]
list_2 = ['Manfred', 'Stefan', 'Lisa', 'Hanna',]

list_1.reverse()
list_2.reverse()
print(list_1) #from bottom to top
print(list_2)

[23, 20, 12, 90]
['Hanna', 'Lisa', 'Stefan', 'Manfred']
```

In [ ]:

# Tuples

- Immutable in python
- Faster than lists and protect unwanted change
- Same rules like with lists regarding functions, indexing & methods
- Single value tuple still comma need

## Tuples in python

```
In [32]: tuple_1 = ("tuples", "are", "awesome")
tuple_2 = ("still", "great")
single_tuple = (1,) # we need the comma here
print(tuple_1)
print(tuple_2)
print(single_tuple)

('tuples', 'are', 'awesome')
('still', 'great')
(1,)
```

what happens if we try to change a value

```
In [26]: tuple_1 = ("tuples", "are", "awesome")
tuple_2 = ("still", "great")
tuple_1[1] = "were"
print(tuple_1)
print(tuple_2)
```

```
-----  
TypeError                                         Traceback (most recent call last)  
<ipython-input-26-35804d59ebb5> in <module>  
      1 tuple_1 = ("tuples", "are", "awesome")  
      2 tuple_2 = ("still", "great")  
----> 3 tuple_1[1] = "were"  
      4 print(tuple_1)  
      5 print(tuple_2)
```

TypeError: 'tuple' object does not support item assignment

# Tuples

ok, and what about this...

```
In [27]: tuple_1 = ("tuples", "are", "awesome")
tuple_2 = ("still", "great")
tuple_2 = tuple_1
print(tuple_1)
print(tuple_2)
```

```
('tuples', 'are', 'awesome')
('tuples', 'are', 'awesome')
```

so, tuples are immutable, but we still can extend the tuple content with a trick...

```
In [28]: tuple_1 = ("tuples", "are", "awesome")
tuple_2 = ("still", "great")
tuple_2 = tuple_1 + tuple_2
print(tuple_1)
print(tuple_2)
```

```
('tuples', 'are', 'awesome')
('tuples', 'are', 'awesome', 'still', 'great')
```

# Dictionaries

- Like unordered Hash-table with keys & values
- Keys must be immutable
- Thus, dictionaries are mutable
- Access values with key indexing

Functions & Methods	Explanation
<code>dict1.keys()</code>	Returns list of keys
<code>dict1.values()</code>	Returns list of values
<code>dict1.items()</code>	Returns list of key, value tuple pairs
<code>dict1.get(key, default=None)</code>	Returns value of key, if there is none, then default
Key in dict_1	Returns True if key exists, False otherwise
<code>dict1.update(dict2)</code>	Adds dict2 keys & values to dict1
<code>dict1.clear()</code>	Removes all elements from dict
<code>len(dict1)</code>	Total number of keys, values

# Dictionary Operations

## Dictionaries in python

```
In [44]: dict_1 = {"item 1": 5, "item 2": 3, "item 3": 10}
dict_2 = {1: "Tomas", 2: "Sandra", 3: "Luca"}

print(dict_1)
print(dict_1.keys())
print(dict_2.values())

{'item 1': 5, 'item 2': 3, 'item 3': 10}
dict_keys(['item 1', 'item 2', 'item 3'])
dict_values(['Tomas', 'Sandra', 'Luca'])
```

### what about the other methods

```
In [52]: dict_1 = {"item 1": 5, "item 2": 3, "item 3": 10}
dict_2 = {1: "Tomas", 2: "Sandra", 3: "Luca"}
dict_3 = {"Apple": 2.5, "Melon": 1.5, "Bag": 0.5}
condition = dict_1.get("Marvel", None)
a = len(dict_1)
b = 1 in dict_1
dict_1.update(dict_3)

print(condition)
print(a)
print(b)
print(dict_1)

None
3
False
{'item 1': 5, 'item 2': 3, 'item 3': 10, 'Apple': 2.5, 'Melon': 1.5, 'Bag': 0.5}
```

Ok, ok,...  
there is one thing  
which we have to  
mention...

```
78     .ltrim(preg_replace('/\\\\\\\\/', '/', $image_src), '/'))  
79     $_SESSION['_CAPTCHA']['config'] = serialize($captcha_config);  
80  
81     return array(  
82         'code' => $captcha_config['code'],  
83         'image_src' => $image_src  
84     );  
85 }  
86  
87  
88 if( !function_exists('hex2rgb') ) {  
89     function hex2rgb($hex_str, $return_string = false, $separator = ',') {  
90         $hex_str = preg_replace("/[^0-9A-Fa-f]/", '', $hex_str); // Gets a proper hex  
91         $rgb_array = array();  
92         if( strlen($hex_str) == 6 ) {  
93             $color_val = hexdec($hex_str);  
94             $rgb_array['r'] = 0xFF & ($color_val >> 0x10);  
95             $rgb_array['g'] = 0xFF & ($color_val >> 0x8);  
96             $rgb_array['b'] = 0xFF & $color_val;  
97         } elseif( strlen($hex_str) == 3 ) {  
98             $rgb_array['r'] = hexdec(str_repeat(substr($hex_str, 0, 1), 2));  
99             $rgb_array['g'] = hexdec(str_repeat(substr($hex_str, 1, 1), 2));  
100            $rgb_array['b'] = hexdec(str_repeat(substr($hex_str, 2, 1), 2));  
101        } else {  
102            return false;  
103        }  
104    }  
105    return $return_string ? implode($separator, $rgb_array) : $rgb_array;  
106 }  
107  
108 Draw the image  
isset($_GET['action'])  
if( $action == 'draw' ) {  
    $image_src = hex2rgb($hex_code);  
    $image_src = imagecreatefromstring($image_src);  
    imagepng($image_src);  
    imagedestroy($image_src);  
}  
109  
110 }  
111  
112 }  
113  
114 }  
115  
116 }  
117  
118 }  
119  
120 }  
121  
122 }  
123  
124 }  
125  
126 }  
127  
128 }  
129  
130 }  
131  
132 }  
133  
134 }  
135  
136 }  
137  
138 }  
139  
140 }  
141  
142 }  
143  
144 }  
145  
146 }  
147  
148 }  
149  
150 }  
151  
152 }  
153  
154 }  
155  
156 }  
157  
158 }  
159  
160 }  
161  
162 }  
163  
164 }  
165  
166 }  
167  
168 }  
169  
170 }  
171  
172 }  
173  
174 }  
175  
176 }  
177  
178 }  
179  
180 }  
181  
182 }  
183  
184 }  
185  
186 }  
187  
188 }  
189  
190 }  
191  
192 }  
193  
194 }  
195  
196 }  
197  
198 }  
199  
200 }  
201  
202 }  
203  
204 }  
205  
206 }  
207  
208 }  
209  
210 }  
211  
212 }  
213  
214 }  
215  
216 }  
217  
218 }  
219  
220 }  
221  
222 }  
223  
224 }  
225  
226 }  
227  
228 }  
229  
230 }  
231  
232 }  
233  
234 }  
235  
236 }  
237  
238 }  
239  
240 }  
241  
242 }  
243  
244 }  
245  
246 }  
247  
248 }  
249  
250 }  
251  
252 }  
253  
254 }  
255  
256 }  
257  
258 }  
259  
260 }  
261  
262 }  
263  
264 }  
265  
266 }  
267  
268 }  
269  
270 }  
271  
272 }  
273  
274 }  
275  
276 }  
277  
278 }  
279  
280 }  
281  
282 }  
283  
284 }  
285  
286 }  
287  
288 }  
289  
290 }  
291  
292 }  
293  
294 }  
295  
296 }  
297  
298 }  
299  
300 }  
301  
302 }  
303  
304 }  
305  
306 }  
307  
308 }  
309  
310 }  
311  
312 }  
313  
314 }  
315  
316 }  
317  
318 }  
319  
320 }  
321  
322 }  
323  
324 }  
325  
326 }  
327  
328 }  
329  
330 }  
331  
332 }  
333  
334 }  
335  
336 }  
337  
338 }  
339  
340 }  
341  
342 }  
343  
344 }  
345  
346 }  
347  
348 }  
349  
350 }  
351  
352 }  
353  
354 }  
355  
356 }  
357  
358 }  
359  
360 }  
361  
362 }  
363  
364 }  
365  
366 }  
367  
368 }  
369  
370 }  
371  
372 }  
373  
374 }  
375  
376 }  
377  
378 }  
379  
380 }  
381  
382 }  
383  
384 }  
385  
386 }  
387  
388 }  
389  
390 }  
391  
392 }  
393  
394 }  
395  
396 }  
397  
398 }  
399  
400 }  
401  
402 }  
403  
404 }  
405  
406 }  
407  
408 }  
409  
410 }  
411  
412 }  
413  
414 }  
415  
416 }  
417  
418 }  
419  
420 }  
421  
422 }  
423  
424 }  
425  
426 }  
427  
428 }  
429  
430 }  
431  
432 }  
433  
434 }  
435  
436 }  
437  
438 }  
439  
440 }  
441  
442 }  
443  
444 }  
445  
446 }  
447  
448 }  
449  
450 }  
451  
452 }  
453  
454 }  
455  
456 }  
457  
458 }  
459  
460 }  
461  
462 }  
463  
464 }  
465  
466 }  
467  
468 }  
469  
470 }  
471  
472 }  
473  
474 }  
475  
476 }  
477  
478 }  
479  
480 }  
481  
482 }  
483  
484 }  
485  
486 }  
487  
488 }  
489  
490 }  
491  
492 }  
493  
494 }  
495  
496 }  
497  
498 }  
499  
500 }  
501  
502 }  
503  
504 }  
505  
506 }  
507  
508 }  
509  
510 }  
511  
512 }  
513  
514 }  
515  
516 }  
517  
518 }  
519  
520 }  
521  
522 }  
523  
524 }  
525  
526 }  
527  
528 }  
529  
530 }  
531  
532 }  
533  
534 }  
535  
536 }  
537  
538 }  
539  
540 }  
541  
542 }  
543  
544 }  
545  
546 }  
547  
548 }  
549  
550 }  
551  
552 }  
553  
554 }  
555  
556 }  
557  
558 }  
559  
560 }  
561  
562 }  
563  
564 }  
565  
566 }  
567  
568 }  
569  
570 }  
571  
572 }  
573  
574 }  
575  
576 }  
577  
578 }  
579  
580 }  
581  
582 }  
583  
584 }  
585  
586 }  
587  
588 }  
589  
590 }  
591  
592 }  
593  
594 }  
595  
596 }  
597  
598 }  
599  
600 }  
601  
602 }  
603  
604 }  
605  
606 }  
607  
608 }  
609  
610 }  
611  
612 }  
613  
614 }  
615  
616 }  
617  
618 }  
619  
620 }  
621  
622 }  
623  
624 }  
625  
626 }  
627  
628 }  
629  
630 }  
631  
632 }  
633  
634 }  
635  
636 }  
637  
638 }  
639  
640 }  
641  
642 }  
643  
644 }  
645  
646 }  
647  
648 }  
649  
650 }  
651  
652 }  
653  
654 }  
655  
656 }  
657  
658 }  
659  
660 }  
661  
662 }  
663  
664 }  
665  
666 }  
667  
668 }  
669  
670 }  
671  
672 }  
673  
674 }  
675  
676 }  
677  
678 }  
679  
680 }  
681  
682 }  
683  
684 }  
685  
686 }  
687  
688 }  
689  
690 }  
691  
692 }  
693  
694 }  
695  
696 }  
697  
698 }  
699  
700 }  
701  
702 }  
703  
704 }  
705  
706 }  
707  
708 }  
709  
710 }  
711  
712 }  
713  
714 }  
715  
716 }  
717  
718 }  
719  
720 }  
721  
722 }  
723  
724 }  
725  
726 }  
727  
728 }  
729  
730 }  
731  
732 }  
733  
734 }  
735  
736 }  
737  
738 }  
739  
740 }  
741  
742 }  
743  
744 }  
745  
746 }  
747  
748 }  
749  
750 }  
751  
752 }  
753  
754 }  
755  
756 }  
757  
758 }  
759  
760 }  
761  
762 }  
763  
764 }  
765  
766 }  
767  
768 }  
769  
770 }  
771  
772 }  
773  
774 }  
775  
776 }  
777  
778 }  
779  
780 }  
781  
782 }  
783  
784 }  
785  
786 }  
787  
788 }  
789  
790 }  
791  
792 }  
793  
794 }  
795  
796 }  
797  
798 }  
799  
800 }  
801  
802 }  
803  
804 }  
805  
806 }  
807  
808 }  
809  
810 }  
811  
812 }  
813  
814 }  
815  
816 }  
817  
818 }  
819  
820 }  
821  
822 }  
823  
824 }  
825  
826 }  
827  
828 }  
829  
830 }  
831  
832 }  
833  
834 }  
835  
836 }  
837  
838 }  
839  
840 }  
841  
842 }  
843  
844 }  
845  
846 }  
847  
848 }  
849  
850 }  
851  
852 }  
853  
854 }  
855  
856 }  
857  
858 }  
859  
860 }  
861  
862 }  
863  
864 }  
865  
866 }  
867  
868 }  
869  
870 }  
871  
872 }  
873  
874 }  
875  
876 }  
877  
878 }  
879  
880 }  
881  
882 }  
883  
884 }  
885  
886 }  
887  
888 }  
889  
890 }  
891  
892 }  
893  
894 }  
895  
896 }  
897  
898 }  
899  
900 }  
901  
902 }  
903  
904 }  
905  
906 }  
907  
908 }  
909  
910 }  
911  
912 }  
913  
914 }  
915  
916 }  
917  
918 }  
919  
920 }  
921  
922 }  
923  
924 }  
925  
926 }  
927  
928 }  
929  
930 }  
931  
932 }  
933  
934 }  
935  
936 }  
937  
938 }  
939  
940 }  
941  
942 }  
943  
944 }  
945  
946 }  
947  
948 }  
949  
950 }  
951  
952 }  
953  
954 }  
955  
956 }  
957  
958 }  
959  
960 }  
961  
962 }  
963  
964 }  
965  
966 }  
967  
968 }  
969  
970 }  
971  
972 }  
973  
974 }  
975  
976 }  
977  
978 }  
979  
980 }  
981  
982 }  
983  
984 }  
985  
986 }  
987  
988 }  
989  
990 }  
991  
992 }  
993  
994 }  
995  
996 }  
997  
998 }  
999  
999 }
```

# Mutable vs. Immutable

## Difference between mutable and immutable

Everything in Python is an object, so every variable holds an object instance. When an object is initiated, it is assigned a unique object id (it is like the memory address). Its type is defined at runtime and once set can never change, however its state can be changed if it is mutable. Simple put, a mutable object can be changed after it is created, and an immutable object can't.

In [8]:

```
x = 10  
y = 10  
  
print(id(x))  
print(id(y))
```

```
1666111808  
1666111808
```

In [11]:

```
x = x + 1  
  
print(id(x))  
print(id(y))  
print("So there is a change of address, I did not change the original object!")
```

```
1666111856  
1666111808
```

So there is a change of address

In [26]:

```
m = [1,2,3,4,5,6,7,8]  
n = m  
print(id(m))  
print(id(n))
```

```
111869544  
111869544
```

In [29]:

```
print(m.pop())  
print(id(m))  
print(id(n))  
print("I did change the original object!")  
print(m)
```

```
6  
111869544  
111869544
```

```
I did change the original object!  
[1, 2, 3, 4, 5]
```

# Mutable vs. Immutable II

## Okay, there is another thing to remember...

We know that tuples are immutable. That means value of a tuple can't be changed after it is created. But the "value" of a tuple is in fact a sequence of names with unchangeable bindings to objects. The key thing to note is that the bindings are unchangeable, not the objects they are bound to.

```
In [44]: t1 = ('holberton', [1, 2, 3])
print(t1)
print(id(t1), '\n')

t2 = ('amazing', 'spiderman')
print(t2)
print(id(t2), '\n')

t1 = t1 + t2
print(t1)
print(id(t1)) # So there are three different objects now

# But what if I try to change the value of the List
t1[1].append('A New World!')

print(t1)
print(id(t1), '\n')

# But what if I try to change the value of the holberton
t1[0] = ('Wubba lubba dub dub')

print(t1)
print(id(t1), '\n')

('holberton', [1, 2, 3])
95508240

('amazing', 'spiderman')
95514136

('holberton', [1, 2, 3], 'amazing', 'spiderman')
117088096

('holberton', [1, 2, 3, 'A New World!'], 'amazing', 'spiderman')
117088096
```

## Okay, what about functions...

A mutable object is called by reference in a function, this means that it can change the original variable itself. In order to avoid this, the original variable needs to be copied to another variable. Immutable objects can be called by reference because its value cannot be changed anyways.

```
In [51]: def updateList(list):
    list += [10]
    n = [5, 6]
    print(id(n), '\n')
    updateList(n)
    print(id(n))
    print(id(n), '\n')
    print(id(n))

2068396
[5, 6, 10]
2068396

In [52]: def updateNumber(a):
    print(id(a), '\n')
    n = 18
    print(id(a), '\n')
    updateNumber(a)
    print(id(a))
    print(a)

b = 8
print(id(b), '\n')
updateNumber(b)
print(id(b))
print(b)

1666111776
1666111776
1666111936
8
```

```
-----  
TypeError                                         Traceback (most recent call last)  
<ipython-input-44-5af3c04fd624> in <module>  
      18  
      19 # But what if I try to change the value of the holberton  
----> 20 t1[0] = ('Wubba lubba dub dub')  
      21  
      22 print(t1)  
  
TypeError: 'tuple' object does not support item assignment
```

# Mutable vs. Immutable III

## Okay, what about functions...

A mutable object is called by reference in a function, this means that it can change the original variable itself. In order to avoid this, the original variable needs to be copied to another variable. Immutable objects can be called by reference because its value cannot be changed anyways.

```
In [55]: def updateList(list1):
    list1 += [10]

    n = [5, 6]
    print(id(n), '\n')           # 26063896
    updateList(n)
    print(n, '\n')              # [5, 6, 10]
    print(id(n), '\n')          # 26063896
```

26063896

[5, 6, 10]

26063896

```
In [52]: def updateNumber(n):
    print(id(n), '\n')
    n += 10
    print(id(n), '\n') # 1666111936
```

```
b = 8
print(id(b), '\n') # 1666111728
updateNumber(b)   # 1666111728
print(b)          # 8
```

1666111776

1666111776

1666111936

# Challenge 2



One final thing...*always*  
*look on the bride side of life*



# Thanks!

## Python Team

**Wiki:**

<https://wiki.tum.de/display/ldv/Info>

**Mail:**

[pythonworkshop.tum@gmail.com](mailto:pythonworkshop.tum@gmail.com)

**Web:**

<https://www.ei.tum.de/startseite/>

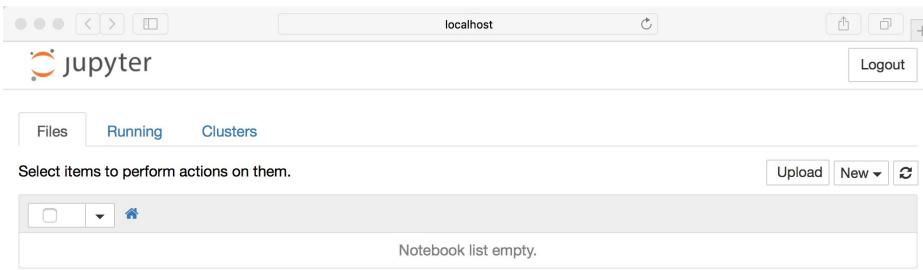
**Git:**

<https://gitlab.ldv.ei.tum.de/daedalus/python>

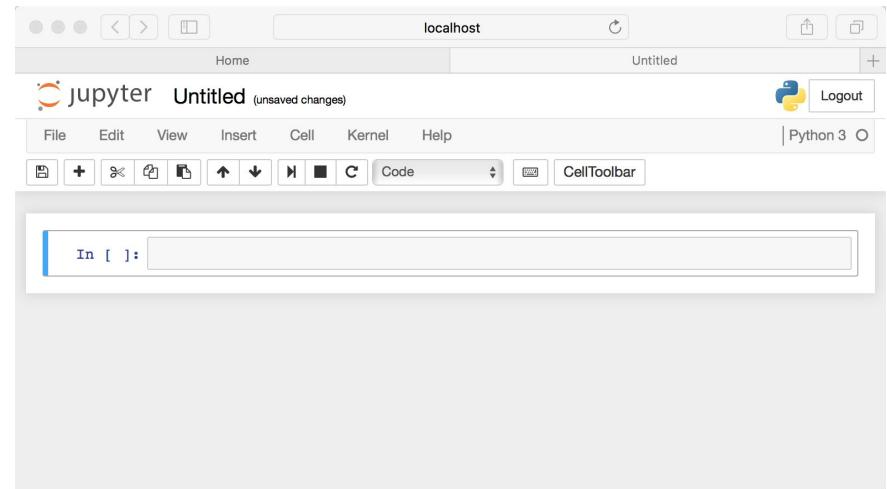


# Backup - Jupyter Notebook - Overview

## Main Menu:



## Creating a Notebook



# Backup - Jupyter Notebook - Overview

## Edit Mode:

jupyter Test Notebook Last Checkpoint: 17/04/2019 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Trusted

# Conda - package manager

```
_let's have a look at conda -->
```

## Cloning an environment

In [ ]: conda create --name myclone --clone myenv # this will create an environment with the name myclone

## Activate / Deactivate an environment

```
You have your virtual environment created! Now, in order to work with it, you have to __activate__ it! Type in your command prompt / terminal...  
btw. your default environment is __base__
```

## Command Mode:

jupyter Test Notebook Last Checkpoint: 17/04/2019 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Trusted

# Conda - package manager

```
_let's have a look at conda -->
```

## Cloning an environment

In [ ]: conda create --name myclone --clone myenv # this will create an environment with the name myclone

## Activate / Deactivate an environment

```
You have your virtual environment created! Now, in order to work with it, you have to __activate__ it! Type in your command prompt / terminal...  
btw. your default environment is __base__
```

In [ ]: conda activate virtualenv # this will activate your virtual environment call virtualenv

# Backup - Jupyter Notebook - Overview

## Run a cell:

A screenshot of a Jupyter Notebook interface. The title bar says "jupyter Hello Jupyter Last Checkpoint: Yesterday at 8:53 AM (unsaved changes)". The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. Below the menu is a toolbar with icons for file operations like Open, Save, and Run. The main area shows two code cells. The first cell, labeled "In [2]:", contains the Python code `print('Hello Jupyter')`. The output of this cell, "Hello Jupyter", is displayed below it. A second cell, labeled "In [ ]:", is currently selected and empty.

## Change kernel:

A screenshot of a Jupyter Notebook interface titled "pyter Test Notebook Last Checkpoint: 17/04/2019 (autosaved)". The menu bar includes Edit, View, Insert, Cell, Kernel, Widgets, and Help. The "Kernel" menu is open, showing options: Interrupt, Restart, Restart & Clear Output, Restart & Run All, Reconnect, Shutdown, Change kernel, Conda Packages, and Visit anaconda.org. A submenu for "Change kernel" is open, listing "Python 3", "test2", and "testenv". The main notebook area shows a cell with "# Conda - pa" and another cell with "## Cloning ane" and "conda create --name nv # this will create an environment with".

# Backup - Jupyter Notebook - Overview

## Export a Notebook:

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** jupyter Test Notebook Last Checkpoint: 17/04/2019 (autosaved)
- File Menu:** New Notebook, Open..., Make a Copy..., Save as..., Rename..., Save and Checkpoint, Revert to Checkpoint, Print Preview, Download as (Notebook (.ipynb), Python (.py), HTML (.html), Reveal.js slides (.html), Markdown (.md), reST (.rst), LaTeX (.tex), PDF via LaTeX (.pdf), asciidoc (.asciidoc)), Trusted Notebook, Close and Halt.
- Code Cells:**
  - In [ ]: conda - package manager
  - In [ ]: conda have a look at conda -->
  - In [ ]: cloning an environment
  - In [ ]: create --name myclone --clone myenv # this will create an environment
  - In [ ]: activate / Deactivate an environment
  - In [ ]: environment created! Now, in order to work with ...
  - In [ ]: environment is \_\_base\_\_
  - In [ ]: this will activate your virtual environment
  - In [ ]: will activate your virtual environment
- Bottom Status Bar:** In [ ]: conda

# Backup - Jupyter Notebook - Overview

Just press:  
“H”  
on your keyboard

## Shortcuts:

### Keyboard shortcuts

The Jupyter Notebook has two different keyboard input modes. **Edit mode** allows you to type code or text into a cell and is indicated by a green cell border. **Command mode** binds the keyboard to notebook level commands and is indicated by a grey cell border with a blue left margin.

#### Mac OS X modifier keys:

⌘ : Command  
⌃ : Control  
⌥ : Option

⇧ : Shift  
⏎ : Return  
␣ : Space  
⇥ : Tab

#### Command Mode (press `Esc` to enable)

⌃ ⌘ F : find and replace  
⌃ ⌘ E : enter edit mode  
⌃ ⌘ F : open the command palette  
⌃ ⌘ P : open the command palette  
⌃ ⌘ P : open the command palette  
⌃ ⌘ F : open the command palette  
⌃ ⌘ ⌘ ⌘ : run cell, select below  
⌃ ⌘ ⌘ ⌘ : run selected cells  
⌃ ⌘ ⌘ ⌘ : run cell and insert below  
⌃ ⌘ ⌘ ⌘ : change cell to code

#### Edit Shortcuts

⌃ ⌘ ⌘ ⌘ ⌘ : extend selected cells below  
⌃ ⌘ ⌘ ⌘ ⌘ : extend selected cells below  
⌃ A : insert cell above  
⌃ B : insert cell below  
⌃ X : cut selected cells  
⌃ C : copy selected cells  
⌃ V : paste cells above  
⌃ V : paste cells below  
⌃ Z : undo cell deletion

Close