

# Python Cheat Sheet - Keywords

“A puzzle a day to learn, code, and play” → Visit [finxter.com](https://finxter.com)

Keyword	Description	Code example
<code>False, True</code>	Data values from the data type Boolean	<code>False == (1 &gt; 2), True == (2 &gt; 1)</code>
<code>and, or, not</code>	Logical operators: ( <code>x and y</code> ) → both x and y must be True ( <code>x or y</code> ) → either x or y must be True ( <code>not x</code> ) → x must be false	<code>x, y = True, False</code> <code>(x or y) == True</code> # True <code>(x and y) == False</code> # True <code>(not y) == True</code> # True
<code>break</code>	Ends loop prematurely	<code>while(True):</code> <code>break</code> # no infinite loop <code>print("hello world")</code>
<code>continue</code>	Finishes current loop iteration	<code>while(True):</code> <code>continue</code> <code>print("43")</code> # dead code
<code>class</code>  <code>def</code>	Defines a new class → a real-world concept (object oriented programming)  Defines a new function or class method. For latter, first parameter (“self”) points to the class object. When calling class method, first parameter is implicit.	<code>class Beer:</code> <code>def __init__(self):</code> <code>self.content = 1.0</code> <code>def drink(self):</code> <code>self.content = 0.0</code>  <code>becks = Beer()</code> # constructor - create class <code>becks.drink()</code> # beer empty: <code>b.content == 0</code>
<code>if, elif, else</code>	Conditional program execution: program starts with “if” branch, tries the “elif” branches, and finishes with “else” branch (until one branch evaluates to True).	<code>x = int(input("your value: "))</code> <code>if x &gt; 3: print("Big")</code> <code>elif x == 3: print("Medium")</code> <code>else: print("Small")</code>
<code>for, while</code>	<code># For loop declaration</code> <code>for i in [0,1,2]:</code> <code>print(i)</code>	<code># While loop - same semantics</code> <code>j = 0</code> <code>while j &lt; 3:</code> <code>print(j)</code> <code>j = j + 1</code>
<code>in</code>	Checks whether element is in sequence	<code>42 in [2, 39, 42]</code> # True
<code>is</code>	Checks whether both elements point to the same object	<code>y = x = 3</code> <code>x is y</code> # True <code>[3] is [3]</code> # False
<code>None</code>	Empty value constant	<code>def f():</code> <code>x = 2</code> <code>f() is None</code> # True
<code>lambda</code>	Function with no name (anonymous function)	<code>(lambda x: x + 3)(3)</code> # returns 6
<code>return</code>	Terminates execution of the function and passes the flow of execution to the caller. An optional value after the return keyword specifies the function result.	<code>def incrementor(x):</code> <code>return x + 1</code> <code>incrementor(4)</code> # returns 5

# Python Cheat Sheet - Basic Data Types

“A puzzle a day to learn, code, and play” → Visit [finxter.com](https://finxter.com)

	Description	Example
<b>Boolean</b>	<p>The Boolean data type is a truth value, either <b>True</b> or <b>False</b>.</p> <p>The Boolean operators ordered by priority: <b>not</b> x → “if x is False, then x, else y” x <b>and</b> y → “if x is False, then x, else y” x <b>or</b> y → “if x is False, then y, else x”</p> <p>These comparison operators evaluate to <b>True</b>: <b>1 &lt; 2 and 0 &lt;= 1 and 3 &gt; 2 and 2 &gt;= 2 and 1 == 1 and 1 != 0 # True</b></p>	<pre>## 1. Boolean Operations x, y = True, False print(x and not y) # True print(not x and y or x) # True  ## 2. If condition evaluates to False if None or 0 or 0.0 or '' or [] or {} or set():     # None, 0, 0.0, empty strings, or empty     # container types are evaluated to False print("Dead code") # Not reached</pre>
<b>Integer, Float</b>	<p>An integer is a positive or negative number without floating point (e.g. 3). A float is a positive or negative number with floating point precision (e.g. 3.14159265359).</p> <p>The <code>//</code> operator performs integer division. The result is an integer value that is rounded towards the smaller integer number (e.g. <code>3 // 2 == 1</code>).</p>	<pre>## 3. Arithmetic Operations x, y = 3, 2 print(x + y) # = 5 print(x - y) # = 1 print(x * y) # = 6 print(x / y) # = 1.5 print(x // y) # = 1 print(x % y) # = 1s print(-x) # = -3 print(abs(-x)) # = 3 print(int(3.9)) # = 3 print(float(3)) # = 3.0 print(x ** y) # = 9</pre>
<b>String</b>	<p>Python Strings are sequences of characters.</p> <p>The four main ways to create strings are the following.</p> <ol style="list-style-type: none"><li>1. Single quotes <code>'Yes'</code></li><li>2. Double quotes <code>"Yes"</code></li><li>3. Triple quotes (multi-line) <code>"""Yes We Can"""</code></li><li>4. String method <code>str(5) == '5' # True</code></li><li>5. Concatenation <code>"Ma" + "hatma" # 'Mahatma'</code></li></ol> <p>These are whitespace characters in strings.</p> <ul style="list-style-type: none"><li>• Newline \n</li><li>• Space \s</li><li>• Tab \t</li></ul>	<pre>## 4. Indexing and Slicing s = "The youngest pope was 11 years old" print(s[0]) # 'T' print(s[1:3]) # 'he' print(s[-3:-1]) # 'ol' print(s[-3:]) # 'old' x = s.split() # creates string array of words print(x[-3] + " " + x[-1] + " " + x[2] + "s") # '11 old popes'  ## 5. Most Important String Methods y = " This is lazy\n " print(y.strip()) # Remove Whitespace: 'This is lazy' print("DrDre".lower()) # Lowercase: 'drdre' print("attention".upper()) # Uppercase: 'ATTENTION' print("smartphone".startswith("smart")) # True print("smartphone".endswith("phone")) # True print("another".find("other")) # Match index: 2 print("cheat".replace("ch", "m")) # 'meat' print(', '.join(["F", "B", "I"])) # 'F,B,I' print(len("Rumpelstiltskin")) # String length: 15 print("ear" in "earth") # Contains: True</pre>

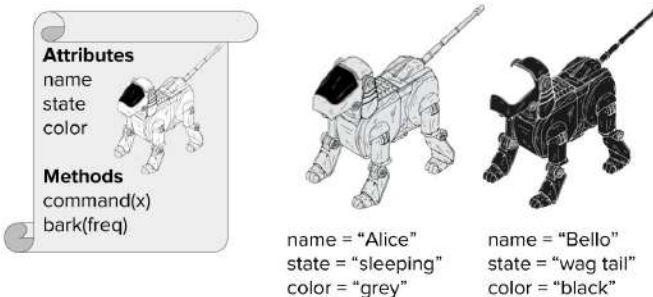
# Python Cheat Sheet - Complex Data Types

“A puzzle a day to learn, code, and play” → Visit [finxter.com](https://finxter.com)

	Description	Example
List	A container data type that stores a sequence of elements. Unlike strings, lists are mutable: modification possible.	<pre>l = [1, 2, 2] print(len(l)) # 3</pre>
Adding elements	Add elements to a list with (i) append, (ii) insert, or (iii) list concatenation. The append operation is very fast.	<pre>[1, 2, 2].append(4) # [1, 2, 2, 4] [1, 2, 4].insert(2,2) # [1, 2, 2, 4] [1, 2, 2] + [4] # [1, 2, 2, 4]</pre>
Removal	Removing an element can be slower.	<pre>[1, 2, 2, 4].remove(1) # [2, 2, 4]</pre>
Reversing	This reverses the order of list elements.	<pre>[1, 2, 3].reverse() # [3, 2, 1]</pre>
Sorting	Sorts a list. The computational complexity of sorting is $O(n \log n)$ for $n$ list elements.	<pre>[2, 4, 2].sort() # [2, 2, 4]</pre>
Indexing	Finds the first occurrence of an element in the list & returns its index. Can be slow as the whole list is traversed.	<pre>[2, 2, 4].index(2) # index of element 4 is "0" [2, 2, 4].index(2,1) # index of element 2 after pos 1 is "1"</pre>
Stack	Python lists can be used intuitively as stack via the two list operations <code>append()</code> and <code>pop()</code> .	<pre>stack = [3] stack.append(42) # [3, 42] stack.pop() # 42 (stack: [3]) stack.pop() # 3 (stack: [])</pre>
Set	A set is an unordered collection of elements. Each can exist only once.	<pre>basket = {'apple', 'eggs', 'banana', 'orange'} same = set(['apple', 'eggs', 'banana', 'orange'])</pre>
Dictionary	The dictionary is a useful data structure for storing (key, value) pairs.	<pre>calories = {'apple' : 52, 'banana' : 89, 'choco' : 546}</pre>
Reading and writing elements	Read and write elements by specifying the key within the brackets. Use the <code>keys()</code> and <code>values()</code> functions to access all keys and values of the dictionary.	<pre>print(calories['apple'] &lt; calories['choco']) # True calories['cappu'] = 74 print(calories['banana'] &lt; calories['cappu']) # False print('apple' in calories.keys()) # True print(52 in calories.values()) # True</pre>
Dictionary Looping	You can loop over the (key, value) pairs of a dictionary with the <code>items()</code> method.	<pre>for k, v in calories.items():     print(k) if v &gt; 500 else None # 'chocolate'</pre>
Membership operator	Check with the 'in' keyword whether the set, list, or dictionary contains an element. Set containment is faster than list containment.	<pre>basket = {'apple', 'eggs', 'banana', 'orange'} print('eggs' in basket) # True print('mushroom' in basket) # False</pre>
List and Set Comprehension	List comprehension is the concise Python way to create lists. Use brackets plus an expression, followed by a for clause. Close with zero or more for or if clauses.  Set comprehension is similar to list comprehension.	<pre># List comprehension l = [('Hi' + x) for x in ['Alice', 'Bob', 'Pete']] print(l) # ['Hi Alice', 'Hi Bob', 'Hi Pete'] l2 = [x * y for x in range(3) for y in range(3) if x&gt;y] print(l2) # [0, 0, 2] # Set comprehension squares = { x**2 for x in [0,2,4] if x &lt; 4 } # {0, 4}</pre>

# Python Cheat Sheet - Classes

“A puzzle a day to learn, code, and play” → Visit [finxter.com](https://finxter.com)

	Description	Example
Classes	<p>A class encapsulates data and functionality - data as attributes, and functionality as methods. It is a blueprint to create concrete instances in the memory.</p> <p><b>Class</b></p>  <p><b>Instances</b></p> <p>name = "Alice" state = "sleeping" color = "grey"</p> <p>name = "Bello" state = "wag tail" color = "black"</p>	<pre>class Dog:     """ Blueprint of a dog """      # class variable shared by all instances     species = ["canis lupus"]      def __init__(self, name, color):         self.name = name         self.state = "sleeping"         self.color = color      def command(self, x):         if x == self.name:             self.bark(2)         elif x == "sit":             self.state = "sit"         else:             self.state = "wag tail"      def bark(self, freq):         for i in range(freq):             print "[" + self.name + ": Woof!"  bello = Dog("bello", "black") alice = Dog("alice", "white")  print(bello.color) # black print(alice.color) # white  bello.bark(1) # [bello]: Woof!  alice.command("sit") print("[alice]: " + alice.state) # [alice]: sit  bello.command("no") print("[bello]: " + bello.state) # [bello]: wag tail  alice.command("alice") # [alice]: Woof! # [alice]: Woof!  bello.species += ["wulf"] print(len(bello.species) == len(alice.species)) # True (!)</pre>
Instance	<p>You are an instance of the class human. An instance is a concrete implementation of a class: all attributes of an instance have a fixed value. Your hair is blond, brown, or black - but never unspecified.</p> <p>Each instance has its own attributes independent of other instances. Yet, class variables are different. These are data values associated with the class, not the instances. Hence, all instance share the same class variable <b>species</b> in the example.</p>	
Self	<p>The first argument when defining any method is always the <b>self</b> argument. This argument specifies the instance on which you call the method.</p> <p><b>self</b> gives the Python interpreter the information about the concrete instance. To <i>define</i> a method, you use <b>self</b> to modify the instance attributes. But to <i>call</i> an instance method, you do not need to specify <b>self</b>.</p>	
Creation	<p>You can create classes “on the fly” and use them as logical units to store complex data types.</p> <pre>class Employee():     pass  employee = Employee() employee.salary = 122000 employee.firstname = "alice" employee.lastname = "wonderland"  print(employee.firstname + " " + employee.lastname + " " + str(employee.salary) + "\$") # alice wonderland 122000\$</pre>	

# Python Cheat Sheet - Functions and Tricks

“A puzzle a day to learn, code, and play” → Visit [finxter.com](https://finxter.com)

		Description	Example	Result
ADVANCED FUNCTIONS	map(func, iter)	Executes the function on all elements of the iterable	list(map(lambda x: x[0], ['red', 'green', 'blue']))	['r', 'g', 'b']
	map(func, i1, ..., ik)	Executes the function on all k elements of the k iterables	list(map(lambda x, y: str(x) + ' ' + y + 's', [0, 2, 2], ['apple', 'orange', 'banana']))	['0 apples', '2 oranges', '2 bananas']
	string.join(iter)	Concatenates iterable elements separated by string	'marries'.join(list(['Alice', 'Bob']))	'Alice marries Bob'
	filter(func, iterable)	Filters out elements in iterable for which function returns False (or 0)	list(filter(lambda x: True if x>17 else False, [1, 15, 17, 18]))	[18]
	string.strip()	Removes leading and trailing whitespaces of string	print("\n\t42\t".strip())	42
	sorted(iter)	Sorts iterable in ascending order	sorted([8, 3, 2, 42, 5])	[2, 3, 5, 8, 42]
	sorted(iter, key=key)	Sorts according to the key function in ascending order	sorted([8, 3, 2, 42, 5], key=lambda x: 0 if x==42 else x)	[42, 2, 3, 5, 8]
	help(func)	Returns documentation of func	help(str.upper())	'... to uppercase.'
	zip(i1, i2, ...)	Groups the i-th elements of iterators i1, i2, ... together	list(zip(['Alice', 'Anna'], ['Bob', 'Jon', 'Frank']))	[('Alice', 'Bob'), ('Anna', 'Jon')]
TRICKS	Unzip	Equal to: 1) unpack the zipped list, 2) zip the result	list(zip(*(['Alice', 'Bob'], ('Anna', 'Jon')))	[('Alice', 'Anna'), ('Bob', 'Jon')]
	enumerate(iter)	Assigns a counter value to each element of the iterable	list(enumerate(['Alice', 'Bob', 'Jon']))	[(0, 'Alice'), (1, 'Bob'), (2, 'Jon')]
	python -m http.server <P>	Share files between PC and phone? Run command in PC's shell. <P> is any port number 0–65535. Type < IP address of PC>:<P> in the phone's browser. You can now browse the files in the PC directory.		
	Read comic	import antigravity	Open the comic series xkcd in your web browser	
	Zen of Python	import this	'...Beautiful is better than ugly. Explicit is ...'	
	Swapping numbers	Swapping variables is a breeze in Python. No offense, Java!	a, b = 'Jane', 'Alice' a, b = b, a	a = 'Alice' b = 'Jane'
	Unpacking arguments	Use a sequence as function arguments via asterisk operator *. Use a dictionary (key, value) via double asterisk operator **	def f(x, y, z): return x + y * z f(*[1, 3, 4]) f(**{'z': 4, 'x': 1, 'y': 3})	13 13
	Extended Unpacking	Use unpacking for multiple assignment feature in Python	a, *b = [1, 2, 3, 4, 5]	a = 1 b = [2, 3, 4, 5]
	Merge two dictionaries	Use unpacking to merge two dictionaries into a single one	x={'Alice': 18} y={'Bob': 27, 'Ann': 22} z = {**x, **y}	z = {'Alice': 18, 'Bob': 27, 'Ann': 22}

# Python Cheat Sheet: 14 Interview Questions

“A puzzle a day to learn, code, and play” →

\*FREE\* Python Email Course @ <http://bit.ly/free-python-course>

Question	Code	Question	Code
Check if list contains integer x	<pre>l = [3, 3, 4, 5, 2, 111, 5] print(111 in l) # True</pre>	Get missing number in [1...100]	<pre>def get_missing_number(lst):     return set(range(lst[len(lst)-1])[1:]) - set(l) l = list(range(1,100)) l.remove(50) print(get_missing_number(l)) # 50</pre>
Find duplicate number in integer list	<pre>def find_duplicates(elements):     duplicates, seen = set(), set()     for element in elements:         if element in seen:             duplicates.add(element)         seen.add(element)     return list(duplicates)</pre>	Compute the intersection of two lists	<pre>def intersect(lst1, lst2):     res, lst2_copy = [], lst2[:]     for el in lst1:         if el in lst2_copy:             res.append(el)             lst2_copy.remove(el)     return res</pre>
Check if two strings are anagrams	<pre>def is_anagram(s1, s2):     return set(s1) == set(s2) print(is_anagram("elvis", "lives")) # True</pre>	Find max and min in unsorted list	<pre>l = [4, 3, 6, 3, 4, 888, 1, -11, 22, 3] print(max(l)) # 888 print(min(l)) # -11</pre>
Remove all duplicates from list	<pre>lst = list(range(10)) + list(range(10)) lst = list(set(lst)) print(lst) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]</pre>	Reverse string using recursion	<pre>def reverse(string):     if len(string)&lt;=1: return string     return reverse(string[1:])+string[0] print(reverse("hello")) # olleh</pre>
Find pairs of integers in list so that their sum is equal to integer x	<pre>def find_pairs(l, x):     pairs = []     for (i, el_1) in enumerate(l):         for (j, el_2) in enumerate(l[i+1:]):             if el_1 + el_2 == x:                 pairs.append((el_1, el_2))     return pairs</pre>	Compute the first n Fibonacci numbers	<pre>a, b = 0, 1 n = 10 for i in range(n):     print(b)     a, b = b, a+b # 1, 1, 2, 3, 5, 8, ...</pre>
Check if a string is a palindrome	<pre>def is_palindrome(phrase):     return phrase == phrase[::-1] print(is_palindrome("anna")) # True</pre>	Sort list with Quicksort algorithm	<pre>def qsort(L):     if L == []: return []     return qsort([x for x in L[1:] if x&lt; L[0]]) + L[0:1] + qsort([x for x in L[1:] if x&gt;=L[0]]) lst = [44, 33, 22, 5, 77, 55, 999] print(qsort(lst)) # [5, 22, 33, 44, 55, 77, 999]</pre>
Use list as stack, array, and queue	<pre># as a list ... l = [3, 4] l += [5, 6] # l = [3, 4, 5, 6]  # ... as a stack ... l.append(10) # l = [4, 5, 6, 10] l.pop() # l = [4, 5, 6]  # ... and as a queue l.insert(0, 5) # l = [5, 4, 5, 6] l.pop() # l = [5, 4, 5]</pre>	Find all permutations of string	<pre>def get_permutations(w):     if len(w)&lt;=1:         return set(w)     smaller = get_permutations(w[1:])     perms = set()     for x in smaller:         for pos in range(0,len(x)+1):             perm = x[:pos] + w[0] + x[pos:]             perms.add(perm)     return perms print(get_permutations("nan")) # {'nna', 'ann', 'nan'}</pre>



# Python Cheat Sheet: NumPy

“A puzzle a day to learn, code, and play” → Visit [finxter.com](https://finxter.com)

Name	Description	Example
<code>a.shape</code>	The shape attribute of NumPy array a keeps a tuple of integers. Each integer describes the number of elements of the axis.	<pre>a = np.array([[1,2],[1,1],[0,0]]) print(np.shape(a))</pre> <code># (3, 2)</code>
<code>a.ndim</code>	The ndim attribute is equal to the length of the shape tuple.	<pre>print(np.ndim(a))</pre> <code># 2</code>
<code>*</code>	The asterisk (star) operator performs the Hadamard product, i.e., multiplies two matrices with equal shape element-wise.	<pre>a = np.array([[2, 0], [0, 2]]) b = np.array([[1, 1], [1, 1]]) print(a*b)</pre> <code># [[2 0] [0 2]]</code>
<code>np.matmul(a,b)</code> , <code>a@b</code>	The standard matrix multiplication operator. Equivalent to the <code>@</code> operator.	<pre>print(np.matmul(a,b))</pre> <code># [[2 2] [2 2]]</code>
<code>np.arange([start, ]stop, [step, ])</code>	Creates a new 1D numpy array with evenly spaced values	<pre>print(np.arange(0,10,2))</pre> <code># [0 2 4 6 8]</code>
<code>np.linspace(start, stop, num=50)</code>	Creates a new 1D numpy array with evenly spread elements within the given interval	<pre>print(np.linspace(0,10,3))</pre> <code># [ 0.  5. 10.]</code>
<code>np.average(a)</code>	Averages over all the values in the numpy array	<pre>a = np.array([[2, 0], [0, 2]]) print(np.average(a))</pre> <code># 1.0</code>
<code>&lt;slice&gt; = &lt;val&gt;</code>	Replace the <code>&lt;slice&gt;</code> as selected by the slicing operator with the value <code>&lt;val&gt;</code> .	<pre>a = np.array([0, 1, 0, 0, 0]) a[::2] = 2 print(a)</pre> <code># [2 1 2 0 2]</code>
<code>np.var(a)</code>	Calculates the variance of a numpy array.	<pre>a = np.array([2, 6]) print(np.var(a))</pre> <code># 4.0</code>
<code>np.std(a)</code>	Calculates the standard deviation of a numpy array	<pre>print(np.std(a))</pre> <code># 2.0</code>
<code>np.diff(a)</code>	Calculates the difference between subsequent values in NumPy array a	<pre>fibs = np.array([0, 1, 1, 2, 3, 5]) print(np.diff(fibs, n=1))</pre> <code># [1 0 1 1 2]</code>
<code>np.cumsum(a)</code>	Calculates the cumulative sum of the elements in NumPy array a.	<pre>print(np.cumsum(np.arange(5)))</pre> <code># [0 1 3 6 10]</code>
<code>np.sort(a)</code>	Creates a new NumPy array with the values from a (ascending).	<pre>a = np.array([10,3,7,1,0]) print(np.sort(a))</pre> <code># [0 1 3 7 10]</code>
<code>np.argsort(a)</code>	Returns the indices of a NumPy array so that the indexed values would be sorted.	<pre>a = np.array([10,3,7,1,0]) print(np.argsort(a))</pre> <code># [4 3 1 2 0]</code>
<code>np.max(a)</code>	Returns the maximal value of NumPy array a.	<pre>a = np.array([10,3,7,1,0]) print(np.max(a))</pre> <code># 10</code>
<code>np.argmax(a)</code>	Returns the index of the element with maximal value in the NumPy array a.	<pre>a = np.array([10,3,7,1,0]) print(np.argmax(a))</pre> <code># 0</code>
<code>np.nonzero(a)</code>	Returns the indices of the nonzero elements in NumPy array a.	<pre>a = np.array([10,3,7,1,0]) print(np.nonzero(a))</pre> <code># [0 1 2 3]</code>



# Python Cheat Sheet: Object Orientation Terms

“A puzzle a day to learn, code, and play” → Visit [finxter.com](https://finxter.com)

	Description	Example
Class	A blueprint to create <b>objects</b> . It defines the data ( <b>attributes</b> ) and functionality ( <b>methods</b> ) of the objects. You can access both attributes and methods via the dot notation.	<pre>class Dog:      # class attribute     is_hairy = True      # constructor     def __init__(self, name):         # instance attribute         self.name = name      # method     def bark(self):         print("Wuff")  bello = Dog("bello") paris = Dog("paris")  print(bello.name) "bello"  print(paris.name) "paris"  class Cat:      # method overloading     def miau(self, times=1):         print("miau " * times)  fifi = Cat()  fifi.miau() "miau "  fifi.miau(5) "miau miau miau miau miau "  # Dynamic attribute fifi.likes = "mice" print(fifi.likes) "mice"  # Inheritance class Persian_Cat(Cat):     classification = "Persian"  mimi = Persian_Cat() print(mimi.miau(3)) "miau miau miau "  print(mimi.classification)</pre>
Object (=instance)	A piece of encapsulated data with functionality in your Python program that is built according to a <b>class</b> definition. Often, an object corresponds to a thing in the real world. An example is the object "Obama" that is created according to the class definition "Person". An object consists of an arbitrary number of <b>attributes</b> and <b>methods</b> , <b>encapsulated</b> within a single unit.	
Instantiation	The process of creating an <b>object</b> of a <b>class</b> . This is done with the constructor method <code>__init__(self, ...)</code> .	
Method	A subset of the overall functionality of an <b>object</b> . The method is defined similarly to a function (using the keyword "def") in the <b>class</b> definition. An object can have an arbitrary number of methods.	
Self	The first argument when defining any method is always the <b>self</b> argument. This argument specifies the <b>instance</b> on which you call the <b>method</b> .  <b>self</b> gives the Python interpreter the information about the concrete instance. To <i>define</i> a method, you use <b>self</b> to modify the instance attributes. But to <i>call</i> an instance method, you do not need to specify <b>self</b> .	
Encapsulation	Binding together data and functionality that manipulates the data.	
Attribute	A variable defined for a class ( <b>class attribute</b> ) or for an object ( <b>instance attribute</b> ). You use attributes to package data into enclosed units (class or instance).	
Class attribute	(=class variable, static variable, static attribute) A variable that is created statically in the <b>class</b> definition and that is shared by all class <b>objects</b> .	
Instance attribute (=instance variable)	A variable that holds data that belongs only to a single instance. Other instances do not share this variable (in contrast to <b>class attributes</b> ). In most cases, you create an instance attribute x in the constructor when creating the instance itself using the self keywords (e.g. <code>self.x = &lt;val&gt;</code> ).	
Dynamic attribute	An <b>instance attribute</b> that is defined dynamically during the execution of the program and that is not defined within any <b>method</b> . For example, you can simply add a new <b>attribute</b> <b>neew</b> to any <b>object</b> <b>o</b> by calling <code>o.neew = &lt;val&gt;</code> .	
Method overloading	You may want to define a method in a way so that there are multiple options to call it. For example for class X, you define a <b>method</b> <code>f(...)</code> that can be called in three ways: <code>f(a)</code> , <code>f(a,b)</code> , or <code>f(a,b,c)</code> . To this end, you can define the method with default parameters (e.g. <code>f(a, b=None, c=None)</code> ).	
Inheritance	<b>Class A</b> can inherit certain characteristics (like <b>attributes</b> or <b>methods</b> ) from class <b>B</b> . For example, the class "Dog" may inherit the attribute "number_of_legs" from the class "Animal". In this case, you would define the inherited class "Dog" as follows: <code>"class Dog(Animal): ..."</code>	



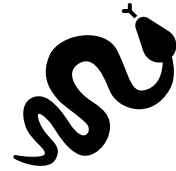
# [Test Sheet] Help Alice Find Her Coding Dad!



+ BONUS



[Solve puzzle 332!](#)



[Solve puzzle 93!](#)



[Solve puzzle 441!](#)



[Solve puzzle 137!](#)



+ BONUS



[Solve puzzle 369!](#)



[Solve puzzle 377!](#)

+ BONUS



[Solve puzzle 366!](#)

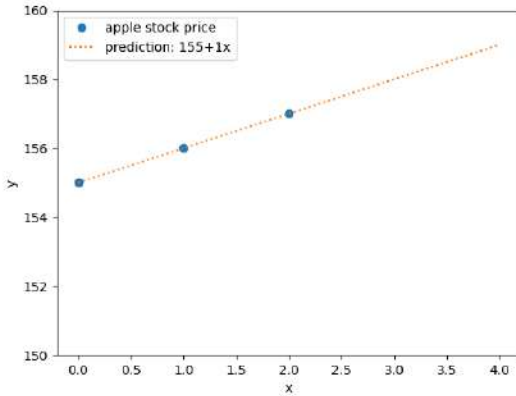


# [Cheat Sheet] 6 Pillar Machine Learning Algorithms

Complete Course: <https://academy.finxter.com/>

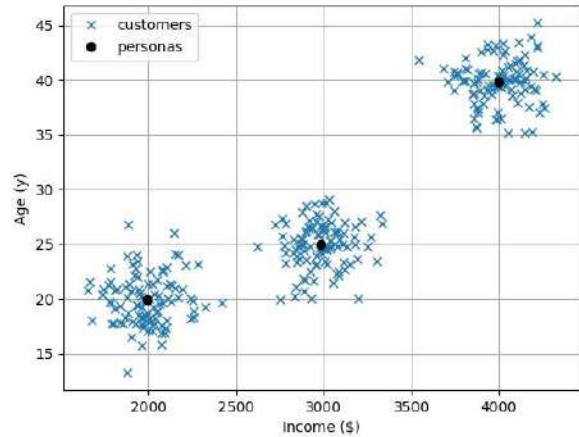
## Linear Regression

<https://blog.finxter.com/logistic-regression-in-one-line-python/>



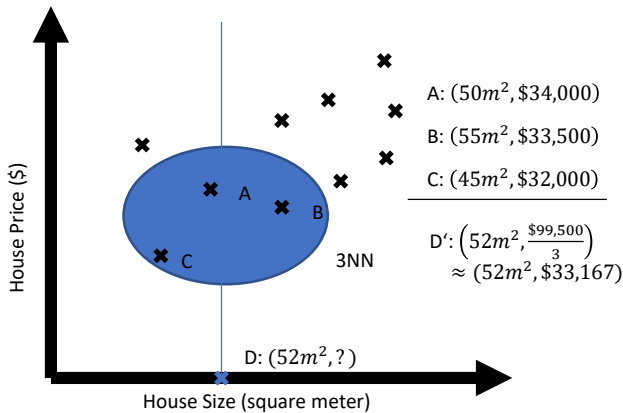
## K-Means Clustering

<https://blog.finxter.com/tutorial-how-to-run-k-means-clustering-in-1-line-of-python/>



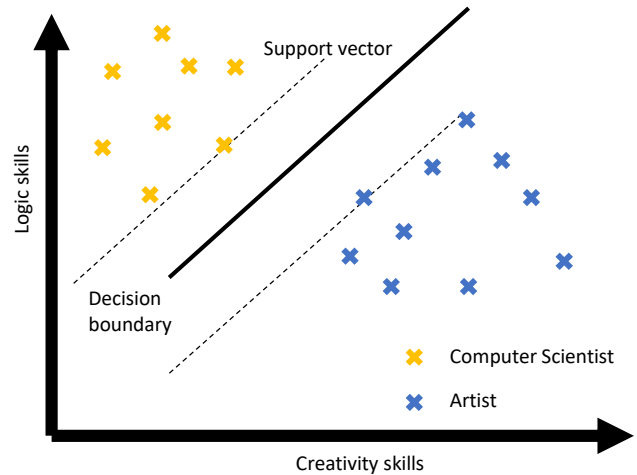
## K Nearest Neighbors

<https://blog.finxter.com/k-nearest-neighbors-as-a-python-one-liner/>



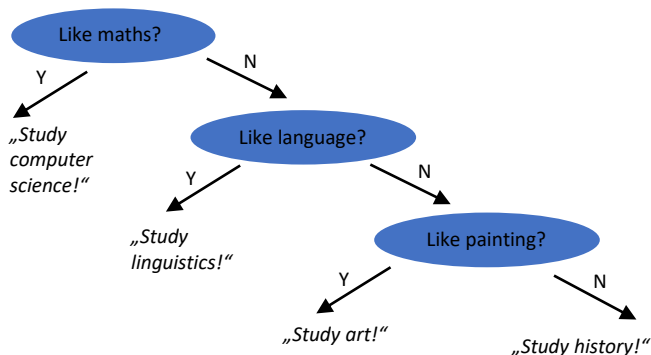
## Support Vector Machine Classification

<https://blog.finxter.com/support-vector-machines-python/>



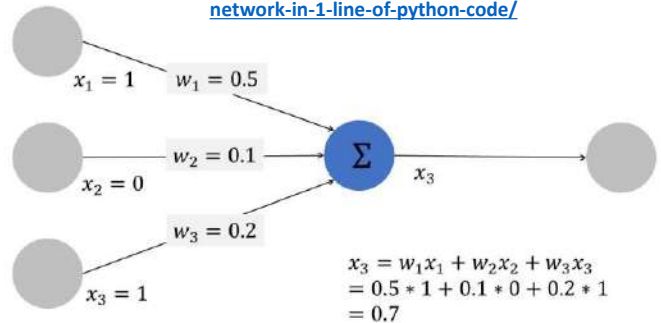
## Decision Tree Classification

<https://blog.finxter.com/decision-tree-learning-in-one-line-python/>



## Multilayer Perceptron

<https://blog.finxter.com/tutorial-how-to-create-your-first-neural-network-in-1-line-of-python-code/>



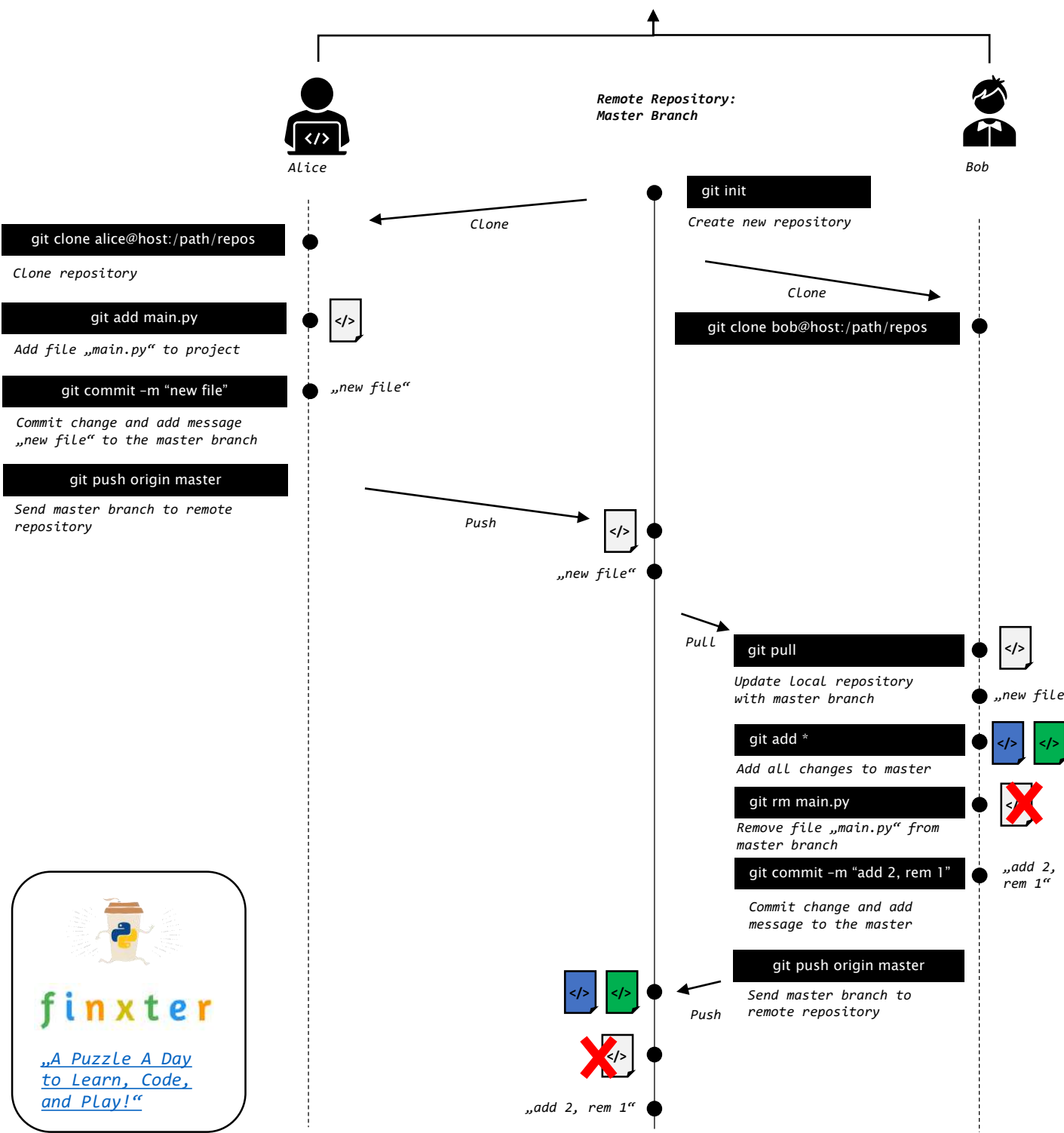
# The Simple Git Cheat Sheet - A Helpful Illustrated Guide

## The Centralized Git Workflow

- Every coder has own copy of project
- Independence of workflow
- No advanced branching and merging needed



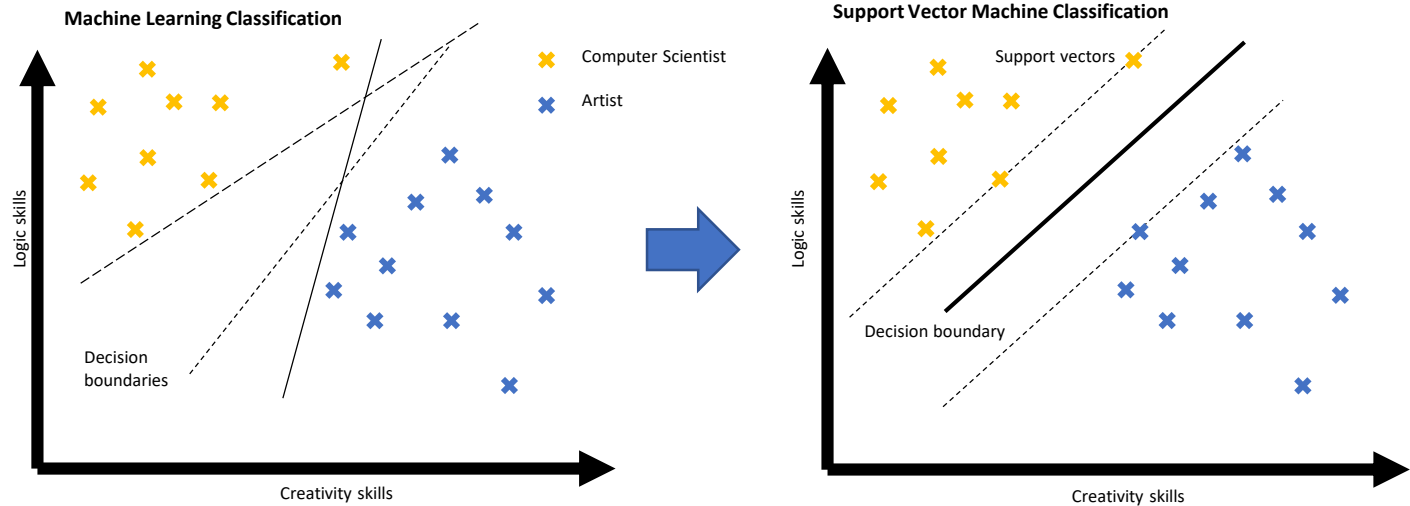
*Git Master Branch*



# [Machine Learning Cheat Sheet] Support Vector Machines

Based on Article: <https://blog.finxter.com/support-vector-machines-python/>

Main idea: Maximize width of separator zone → increases „margin of safety“ for classification



## What are basic SVM properties?

### Support Vector Machines

Alternatives:	SVM, support-vector networks
Learning:	Classification, Regression
Advantages:	Robust for high-dimensional space Memory efficient (only uses support vectors) Flexible and customizable
Disadvantages:	Danger of overfitting in high-dimensional space No classification probabilities like Decision trees
Boundary:	Linear and Non-linear

## What's the explanation of the code example?

### Explanation: A Study Recommendation System with SVM

- NumPy array holds labeled training data (one row per user and one column per feature).
- Features: skill level in maths, language, and creativity.
- Labels: last column is recommended study field.
- 3D data → SVM separates data using 2D planes (the linear separator) rather than 1D lines.
- One-liner:
  1. Create model using constructor of scikit-learn's svm.SVC class (SVC = support vector classification).
  2. Call fit function to perform training based on labeled training data.
- Results: call predict function on new observations
  - student\_0 (skills maths=3, language=3, and creativity=6) → SVM predicts "art"
  - student\_1 (maths=8, language=1, and creativity=1) → SVM predicts "computer science"
- Final output of one-liner:

```
## Dependencies
from sklearn import svm
import numpy as np

## Data: student scores in (maths, language, creativity)
## --> study field
X = np.array([[9, 5, 6, "computer science"],
              [10, 1, 2, "computer science"],
              [1, 8, 1, "literature"],
              [4, 9, 3, "literature"],
              [0, 1, 10, "art"],
              [5, 7, 9, "art"]])

## One-liner
svm = svm.SVC().fit(X[:, :-1], X[:, -1])

## Result & puzzle
student_0 = svm.predict([[3, 3, 6]])
print(student_0)

student_1 = svm.predict([[8, 1, 1]])
print(student_1)
```

```
## Result & puzzle
student_0 = svm.predict([[3, 3, 6]])
print(student_0)
# ['art']

student_1 = svm.predict([[8, 1, 1]])
print(student_1)
## ['computer science']
```






# Python Cheat Sheet: List Methods




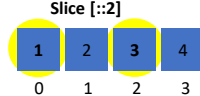
“A puzzle a day to learn, code, and play” → Visit [finxter.com](https://finxter.com)

Method	Description	Example
<code>lst.append(x)</code>	Appends element <code>x</code> to the list <code>lst</code> .	<pre>&gt;&gt;&gt; l = [] &gt;&gt;&gt; l.append(42) &gt;&gt;&gt; l.append(21) [42, 21]</pre>
<code>lst.clear()</code>	Removes all elements from the list <code>lst</code> —which becomes empty.	<pre>&gt;&gt;&gt; lst = [1, 2, 3, 4, 5] &gt;&gt;&gt; lst.clear() []</pre>
<code>lst.copy()</code>	Returns a copy of the list <code>lst</code> . Copies only the list, not the elements in the list (shallow copy).	<pre>&gt;&gt;&gt; lst = [1, 2, 3] &gt;&gt;&gt; lst.copy() [1, 2, 3]</pre>
<code>lst.count(x)</code>	Counts the number of occurrences of element <code>x</code> in the list <code>lst</code> .	<pre>&gt;&gt;&gt; lst = [1, 2, 42, 2, 1, 42, 42] &gt;&gt;&gt; lst.count(42) 3 &gt;&gt;&gt; lst.count(2) 2</pre>
<code>lst.extend(iter)</code>	Adds all elements of an iterable <code>iter</code> (e.g. another list) to the list <code>lst</code> .	<pre>&gt;&gt;&gt; lst = [1, 2, 3] &gt;&gt;&gt; lst.extend([4, 5, 6]) [1, 2, 3, 4, 5, 6]</pre>
<code>lst.index(x)</code>	Returns the position (index) of the first occurrence of value <code>x</code> in the list <code>lst</code> .	<pre>&gt;&gt;&gt; lst = ["Alice", 42, "Bob", 99] &gt;&gt;&gt; lst.index("Alice") 0 &gt;&gt;&gt; lst.index(99, 1, 3) ValueError: 99 is not in list</pre>
<code>lst.insert(i, x)</code>	Inserts element <code>x</code> at position (index) <code>i</code> in the list <code>lst</code> .	<pre>&gt;&gt;&gt; lst = [1, 2, 3, 4] &gt;&gt;&gt; lst.insert(3, 99) [1, 2, 3, 99, 4]</pre>
<code>lst.pop()</code>	Removes and returns the final element of the list <code>lst</code> .	<pre>&gt;&gt;&gt; lst = [1, 2, 3] &gt;&gt;&gt; lst.pop() 3 &gt;&gt;&gt; lst [1, 2]</pre>
<code>lst.remove(x)</code>	Removes and returns the first occurrence of element <code>x</code> in the list <code>lst</code> .	<pre>&gt;&gt;&gt; lst = [1, 2, 99, 4, 99] &gt;&gt;&gt; lst.remove(99) &gt;&gt;&gt; lst [1, 2, 4, 99]</pre>
<code>lst.reverse()</code>	Reverses the order of elements in the list <code>lst</code> .	<pre>&gt;&gt;&gt; lst = [1, 2, 3, 4] &gt;&gt;&gt; lst.reverse() &gt;&gt;&gt; lst [4, 3, 2, 1]</pre>
<code>lst.sort()</code>	Sorts the elements in the list <code>lst</code> in ascending order.	<pre>&gt;&gt;&gt; lst = [88, 12, 42, 11, 2] &gt;&gt;&gt; lst.sort() # [2, 11, 12, 42, 88] &gt;&gt;&gt; lst.sort(key=lambda x: str(x)[0]) # [11, 12, 2, 42, 88]</pre>


## Keywords

Keyword	Description	Code Examples
<code>False</code> , <code>True</code>	Boolean data type	<code>False == (1 &gt; 2)</code> <code>True == (2 &gt; 1)</code> 
<code>and</code> , <code>or</code> , <code>not</code>	Logical operators → Both are true → Either is true → Flips Boolean	<code>True and True</code> # True <code>True or False</code> # True <code>not False</code> # True
<code>break</code>	Ends loop prematurely	<code>while True:</code> <code>break</code> # finite loop
<code>continue</code>	Finishes current loop iteration	<code>while True:</code> <code>continue</code> <code>print("42")</code> # dead code
<code>class</code>	Defines new class	<code>class Coffee:</code> # Define your class
<code>def</code>	Defines a new function or class method.	<code>def say_hi():</code> <code>print('hi')</code>
<code>if</code> , <code>elif</code> , <code>else</code>	Conditional execution: - "if" condition == True? - "elif" condition == True? - Fallback: else branch	<code>x = int(input("ur val:"))</code> <code>if x &gt; 3: print("Big")</code> <code>elif x == 3: print("3")</code> <code>else: print("Small")</code>
<code>for</code> , <code>while</code>	# For loop <code>for i in [0,1,2]:</code> <code>print(i)</code>	# While loop does same <code>j = 0</code> <code>while j &lt; 3:</code> <code>print(j); j = j + 1</code> 
<code>in</code>	Sequence membership	<code>42 in [2, 39, 42]</code> # True
<code>is</code>	Same object memory location	<code>y = x = 3</code> <code>x is y</code> # True <code>[3] is [3]</code> # False
<code>None</code>	Empty value constant	<code>print()</code> is None # True
<code>lambda</code>	Anonymous function	<code>(lambda x: x+3)(3)</code> # 6 
<code>return</code>	Terminates function. Optional return value defines function result.	<code>def increment(x):</code> <code>return x + 1</code> <code>increment(4)</code> # returns 5

## Basic Data Structures

Type	Description	Code Examples
Boolean	The Boolean data type is either <code>True</code> or <code>False</code> . Boolean operators are ordered by priority: <code>not</code> → <code>and</code> → <code>or</code>  <code>{}</code> →  <code>{1, 2, 3}</code> → 	<code>## Evaluates to True:</code> <code>1&lt;2 and 0&lt;=1 and 3&gt;2 and 2&gt;=2 and 1==1 and 1!=0</code>  <code>## Evaluates to False:</code> <code>bool(None or 0 or 0.0 or '' or [] or {} or set())</code>  Rule: <code>None</code> , <code>0</code> , <code>0.0</code> , empty strings, or empty container types evaluate to <code>False</code>
Integer, Float	An integer is a positive or negative number without decimal point such as 3.  A float is a positive or negative number with floating point precision such as 3.1415926.  Integer division rounds toward the smaller integer (example: <code>3//2==1</code> ).	<code>## Arithmetic Operations</code> <code>x, y = 3, 2</code> <code>print(x + y)</code> # 5 <code>print(x - y)</code> # 1 <code>print(x * y)</code> # 6 <code>print(x / y)</code> # 1.5 <code>print(x // y)</code> # 1 <code>print(x % y)</code> # 1 <code>print(-x)</code> # -3 <code>print(abs(-x))</code> # 3 <code>print(int(3.9))</code> # 3 <code>print(float(3))</code> # 3.0 <code>print(x ** y)</code> # 9 
String	Python Strings are sequences of characters.  String Creation Methods: 1. Single quotes >>> 'Yes' 2. Double quotes >>> "Yes" 3. Triple quotes (multi-line) >>> """Yes We Can""" 4. String method >>> str(5) == '5' True 5. Concatenation >>> "Ma" + "hatma" 'Mahatma'  Whitespace chars: Newline \n, Space \s, Tab \t	<code>## Indexing and Slicing</code> <code>s = "The youngest pope was 11 years"</code> <code>s[0]</code> # 'T' <code>s[1:3]</code> # 'he' <code>s[-3:-1]</code> # 'ar' <code>s[-3:]</code> # 'ars'   <code>x = s.split()</code> <code>x[-2] + " " + x[2] + "s" # '11 popes'</code>  <code>## String Methods</code> <code>y = "    Hello world\t\n"</code> <code>y.strip()</code> # Remove Whitespace <code>"HI".lower()</code> # Lowercase: 'hi' <code>"hi".upper()</code> # Uppercase: 'HI' <code>"hello".startswith("he")</code> # True <code>"hello".endswith("lo")</code> # True <code>"hello".find("ll")</code> # Match at 2 <code>"cheat".replace("ch", "m")</code> # 'meat' <code>''.join(["F", "B", "I"])</code> # 'FBI' <code>len("hello world")</code> # Length: 15 <code>"ear" in "earth"</code> # True

## Complex Data Structures

Type	Description	Example
List	Stores a sequence of elements. Unlike strings, you can modify list objects (they're <i>mutable</i> ).	<code>l = [1, 2, 2]</code> <code>print(len(l))</code> # 3 
Adding elements	Add elements to a list with (i) <code>append</code> , (ii) <code>insert</code> , or (iii) list concatenation.	<code>[1, 2].append(4)</code> # [1, 2, 4] <code>[1, 4].insert(1,9)</code> # [1, 9, 4] <code>[1, 2] + [4]</code> # [1, 2, 4]
Removal	Slow for lists	<code>[1, 2, 2, 4].remove(1)</code> # [2, 2, 4]
Reversing	Reverses list order	<code>[1, 2, 3].reverse()</code> # [3, 2, 1]
Sorting	Sorts list using fast Timsort	<code>[2, 4, 2].sort()</code> # [2, 2, 4]
Indexing	Finds the first occurrence of an element & returns index. Slow worst case for whole list traversal.	<code>[2, 2, 4].index(2)</code> # index of item 2 is 0 <code>[2, 2, 4].index(2,1)</code> # index of item 2 after pos 1 is 1
Stack	Use Python lists via the list operations <code>append()</code> and <code>pop()</code>	<code>stack = [3]</code> <code>stack.append(42)</code> # [3, 42] <code>stack.pop()</code> # 42 (stack: [3]) <code>stack.pop()</code> # 3 (stack: [])
Set	An unordered collection of unique elements ( <i>at-most-once</i> ) → fast membership <i>O(1)</i>	<code>basket = {'apple', 'eggs', 'banana', 'orange'}</code> <code>same = set(['apple', 'eggs', 'banana', 'orange'])</code>

Type	Description	Example
Dictionary	Useful data structure for storing (key, value) pairs	<code>cal = {'apple': 52, 'banana': 89, 'choco': 546}</code> # calories
Reading and writing elements	Read and write elements by specifying the key within the brackets. Use the <code>keys()</code> and <code>values()</code> functions to access all keys and values of the dictionary	<code>print(cal['apple'] &lt; cal['choco'])</code> # True <code>cal['cappu'] = 74</code> <code>print(cal['banana'] &lt; cal['cappu'])</code> # False <code>print('apple' in cal.keys())</code> # True <code>print(52 in cal.values())</code> # True
Dictionary Iteration	You can access the (key, value) pairs of a dictionary with the <code>items()</code> method.	<code>for k, v in cal.items():</code> <code>print(k) if v &gt; 500 else ''</code> # 'choco'
Membership operator	Check with the <code>in</code> keyword if set, list, or dictionary contains an element. Set membership is faster than list membership.	<code>basket = {'apple', 'eggs', 'banana', 'orange'}</code> <code>print('eggs' in basket)</code> # True <code>print('mushroom' in basket)</code> # False
List & set comprehension	List comprehension is the concise Python way to create lists. Use brackets plus an expression, followed by a <code>for</code> clause. Close with zero or more <code>for</code> or <code>if</code> clauses. Set comprehension works similar to list comprehension.	<code>l = ['hi' + x for x in ['Alice', 'Bob', 'Pete']]</code> # ['Hi Alice', 'Hi Bob', 'Hi Pete']  <code>l2 = [x * y for x in range(3) for y in range(3) if x&gt;y]</code> # [0, 0, 2]  <code>squares = {x**2 for x in [0,2,4] if x &lt; 4}</code> # {0, 4}



## Complexity

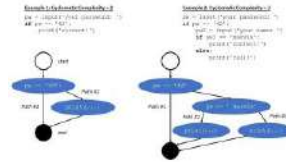
"A whole, made up of parts—difficult to analyze, understand, or explain".

- Complexity appears in
- Project Lifecycle
  - Code Development
  - Algorithmic Theory
  - Processes
  - Social Networks
  - Learning & Your Daily Life

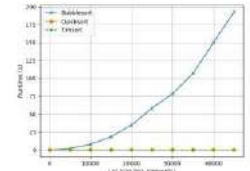
## Project Lifecycle



## Cyclomatic Complexity



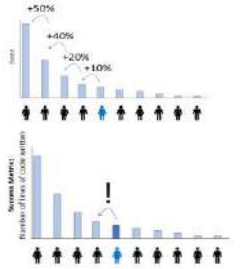
## Runtime Complexity



→ Complexity reduces productivity and focus. It'll consume your precious time. **Keep it simple!**

## 80/20 Principle

Majority of effects come from the minority of causes.



## Pareto Tips

1. Figure out your success metrics.
2. Figure out your big goals in life.
3. Look for ways to achieve the same things with fewer resources.
4. Reflect on your own successes
5. Reflect on your own failures
6. Read more books in your industry.
7. Spend much of your time improving and tweaking existing products
8. Smile.
9. Don't do things that reduce value

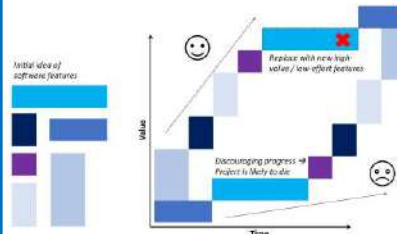
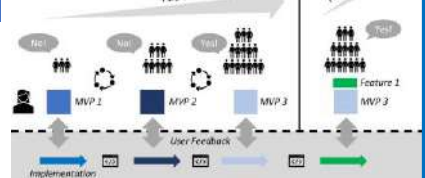
**Maximize Success Metric:**

**#lines of code written**

## Minimum Viable Product (MVP)

A minimum viable product in the software sense is code that is stripped from all features to focus on the core functionality.

## Minimum Viable Product & Iterative Feedback Loop



## How to MVP?

- Formulate hypothesis
- Omit needless features
- Split test to validate each new feature
- Focus on product-market fit
- Seek high-value and low-cost features

## Clean Code Principles

1. You Ain't Going to Need It
2. The Principle of Least Surprise
3. Don't Repeat Yourself
4. **Code For People Not Machines**
5. Stand on the Shoulders of Giants
6. Use the Right Names
7. Single-Responsibility Principle
8. Use Comments
9. Avoid Unnecessary Comments
10. Be Consistent
11. Test
12. Think in Big Pictures
13. Only Talk to Your Friends
14. Refactor
15. Don't Overengineer
16. Don't Overuse Indentation
17. Small is Beautiful
18. Use Metrics
19. Boy Scout Rule: Leave Camp Cleaner Than You Found It

## Unix Philosophy

1. Simple's Better Than Complex
2. **Small is Beautiful (Again)**
3. Make Each Program Do One Thing Well
4. Build a Prototype First
5. Portability Over Efficiency
6. Store Data in Flat Text Files
7. Use Software Leverage
8. Avoid Captive User Interfaces
9. **Program = Filter**
10. Worse is Better
11. Clean > Clever Code
12. **Design Connected Programs**
13. Make Your Code Robust
14. Repair What You Can — But Fail Early and Noisily
15. Write Programs to Write Programs

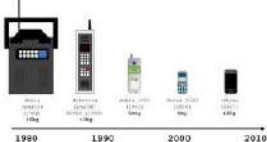
## Premature Optimization

"Programmers waste enormous amounts of time thinking about [...] the speed of noncritical parts of their programs. We should forget about small efficiencies, say about 97% of the time: **premature optimization is the root of all evil.**" — Donald Knuth

## Performance Tuning 101

1. Measure, then improve
2. Focus on the slow 20%
3. Algorithmic optimization wins
4. All hail to the cache
5. Solve an easier problem version
6. Know when to stop

## Less Is More in Design

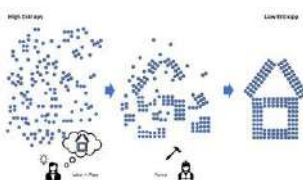


## How to Simplify Design?

1. Use whitespace
2. Remove design elements
3. Remove features
4. Reduce variation of fonts, font types, colors
5. Be consistent across UIs

## Focus

You can take raw resources and move them from a state of high entropy into a state of low entropy—using **focused effort towards the attainment of a greater plan.**



## 3-Step Approach of Efficient Software Creation

1. Plan your code
2. Apply focused effort to make it real.
3. Seek feedback

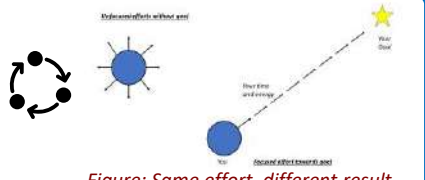


Figure: Same effort, different result.