



How Prompt Engineering Works



Note: Prompt engineering designs and optimizes prompts for language models.

It's important in NLP and language generation. Prompt formats guide the model and can be used for tasks like product descriptions or conversational AI.

Reliable prompt formats exist, but exploring new formats is encouraged.

"{your input here}" is a placeholder for text or context

Rules of Thumb and Examples

Rule #1 – Instructions at beginning and ### or "" to separate instructions or context



Rewrite the text below in more engaging language.
{your input here}



Rewrite the text below in more engaging language.
Text: ""
{your input here}
""

Rule #2 – Be specific and detailed about the desired context, outcome, length, format, and style.



Write a short story for kids



Write a funny soccer story for kids that teaches the kid that persistence is key for success in the style of Rowling.

Rule #3 – Give examples of desired output format



Extract house pricing data from the following text.
Text: ""
{your text containing pricing data}
""



Extract house pricing data from the following text.
Desired format: ""
House 1 | \$1,000,000 | 100 sqm
House 2 | \$500,000 | 90 sqm
... (and so on)
""

Text: ""
{your text containing pricing data}
""

Rule #4 – First try without examples, then try giving some examples.



Extract brand names from the text below.

Text: {your text here}

Brand names:



Extract brand names from the texts below.

Text 1: Finxter and YouTube are tech companies. Google is too.
Brand names 2: Finxter, YouTube, Google
###

Text 2: If you like tech, you'll love Finxter!
Brand names 2: Finxter
###

Text 3: {your text here}
Brand names 3:

Rule #5 – Fine-tune if Rule #4 doesn't work

Fine-tuning improves model performance by training on more examples, resulting in higher quality results, token savings, and lower latency requests.

GPT-3 can intuitively generate plausible completions from few examples, known as **few-shot learning**.

Fine-tuning achieves better results on various tasks without requiring examples in the prompt, saving costs and enabling lower-latency requests.

Example Training Data

```
{"prompt": "<input>", "completion": "<ideal output>"}  
{"prompt": "<input>", "completion": "<ideal output>"}  
{"prompt": "<input>", "completion": "<ideal output>"}  
...
```

Rule #6 – Be specific. Omit needless words.



ChatGPT, write a sales page for my company selling sand in the desert, please write only a few sentences, nothing long and complex



Write a 5-sentence sales page, sell sand in the desert.

Rule #7 – Use leading words to nudge the model towards a pattern



Write a Python function that plots my net worth over 10 years for different inputs on the initial investment and a given ROI



```
# Python function that plots net worth over 10  
# years for different inputs on the initial  
# investment and a given ROI
```

```
import matplotlib
```

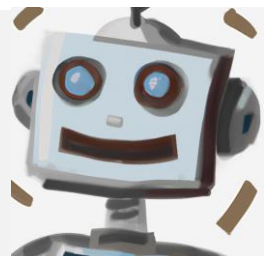
```
def plot_net_worth(initial, roi):
```

Bonus Prompt – Let ChatGPT Design the Optimal Prompt




New models are created frequently, and the performance of subsequent models can be an order of magnitude. You are a robot for creating prompts. You need to gather information about the user's goals, examples of preferred output, and any other relevant contextual information.

The prompt should contain all the necessary information provided to you. Ask the user more questions until you are sure you can create an optimal prompt.




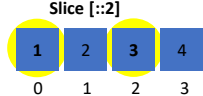
Your answer should be clearly formatted and optimized for ChatGPT interactions. Be sure to start by asking the user about the goals, the desired outcome, and any additional information you may need.




Keywords

Keyword	Description	Code Examples
<code>False</code> , <code>True</code>	Boolean data type	<code>False == (1 > 2)</code> <code>True == (2 > 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 > 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 < 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<2 and 0<=1 and 3>2 and 2>=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 <code>>>> 'Yes'</code> 2. Double quotes <code>>>> "Yes"</code> 3. Triple quotes (multi-line) <code>>>> """Yes</code> We Can""" 4. String method <code>>>> str(5) == '5'</code> True 5. Concatenation <code>>>> "Ma" + "hatma"</code> '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'] < cal['choco'])</code> # True <code>cal['cappu'] = 74</code> <code>print(cal['banana'] < 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 > 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>y]</code> # [0, 0, 2] <code>squares = {x**2 for x in [0,2,4] if x < 4}</code> # {0, 4}



fintxter Book: Simplicity - The Finer Art of Creating Software

Complexity

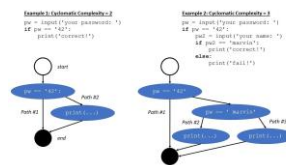
"A whole, made up of parts—difficult to analyze, understand, or explain".

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

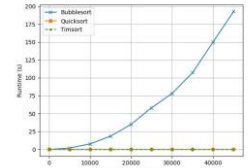
Project Lifecycle



Cyclomatic Complexity



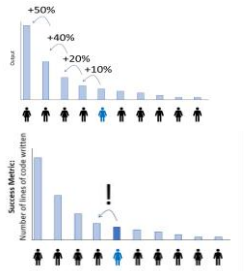
Runtime Complexity



→ 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

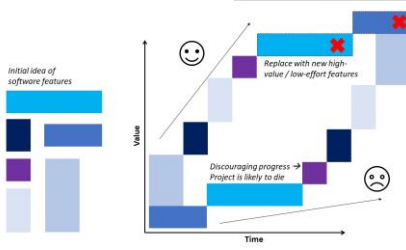
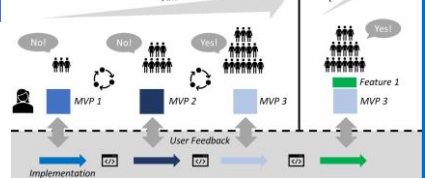
Maximize Success Metric:

#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

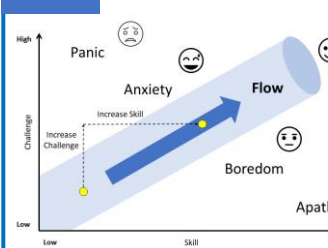


How to Simplify Design?

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

Flow

"... the source code of ultimate human performance" — Kotler



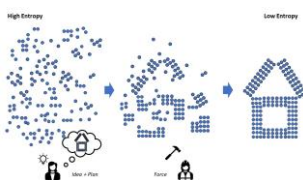
How to Achieve Flow? (1) clear goals, (2) immediate feedback, and (3) balance opportunity & capacity.

Flow Tips for Coders

1. Always work on an explicit practical code project
2. Work on fun projects that fulfill your purpose
3. Perform from your strengths
4. Big chunks of coding time
5. Reduce distractions: smartphone + social
6. Sleep a lot, eat healthily, read quality books, and exercise → garbage in, garbage out!

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

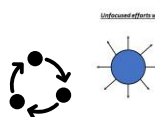


Figure: Same effort, different result.

Subscribe to the **11x FREE** Python Cheat Sheet Course:
<https://blog.fintxter.com/python-cheat-sheets/>

Python Cheat Sheet: Basic Data Types

“A puzzle a day to learn, code, and play” → Visit finxter.com

	Description	Example
Boolean	<p>The Boolean data type is a truth value, either True or False.</p> <p>The Boolean operators ordered by priority: not x → “if x is False, then x, else y” x and y → “if x is False, then x, else y” x or y → “if x is False, then y, else x”</p> <p>These comparison operators evaluate to True: 1 < 2 and 0 <= 1 and 3 > 2 and 2 >= 2 and 1 == 1 and 1 != 0 # True</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>
Integer, Float	<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 // operator performs integer division. The result is an integer value that is rounded toward the smaller integer number (e.g. 3 // 2 == 1).</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>
String	<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">1. Single quotes 'Yes'2. Double quotes "Yes"3. Triple quotes (multi-line) """Yes We Can"""4. String method str(5) == '5' # True5. Concatenation "Ma" + "hatma" # 'Mahatma' <p>These are whitespace characters in strings.</p> <ul style="list-style-type: none">• Newline \n• Space \s• Tab \t	<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\t\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>

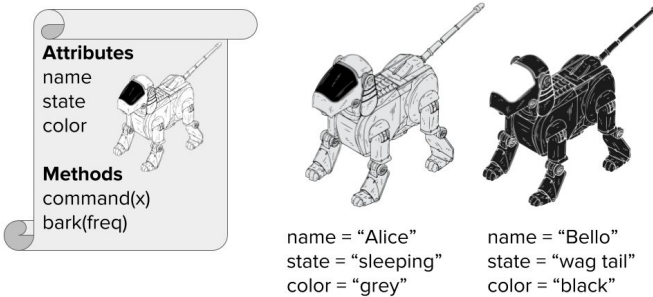
Python Cheat Sheet: Complex Data Types

“A puzzle a day to learn, code, and play” → Visit finxter.com

	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 superlinear in the no. 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 2 is "0" [2, 2, 4].index(2,1) # index of el. 2 after pos 1 is "1"</pre>
Stack	Python lists can be used intuitively as stacks via the two list operations append() and pop().	<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 unique elements (“at-most-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 keys() and values() functions to access all keys and values of the dictionary.	<pre>print(calories['apple'] < calories['choco']) # True calories['cappu'] = 74 print(calories['banana'] < calories['cappu']) # False print('apple' in calories.keys()) # True print(52 in calories.values()) # True</pre>
Dictionary Looping	You can access the (key, value) pairs of a dictionary with the items() method.	<pre>for k, v in calories.items(): print(k) if v > 500 else None # 'choco'</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>y] print(l2) # [0, 0, 2] # Set comprehension squares = { x**2 for x in [0,2,4] if x < 4 } # {0, 4}</pre>

Python Cheat Sheet: Classes

“A puzzle a day to learn, code, and play” → Visit finxter.com

	Description	Example
Classes	<p>A class encapsulates data and functionality: data as attributes, and functionality as methods. It is a blueprint for creating concrete instances in memory.</p> <p>Class Instances</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 species in the example.</p>	
Self	<p>The first argument when defining any method is always the self argument. This argument specifies the instance on which you call the method.</p> <p>self gives the Python interpreter the information about the concrete instance. To <i>define</i> a method, you use self to modify the instance attributes. But to <i>call</i> an instance method, you do not need to specify self.</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: Keywords

“A puzzle a day to learn, code, and play” → Visit finxter.com

Keyword	Description	Code example
<code>False, True</code>	Data values from the data type Boolean	<code>False == (1 > 2), True == (2 > 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	<pre>x, y = True, False (x or y) == True # True (x and y) == False # True (not y) == True # True</pre>
<code>break</code>	Ends loop prematurely	<pre>while(True): break # no infinite loop print("hello world")</pre>
<code>continue</code>	Finishes current loop iteration	<pre>while(True): continue print("43") # dead code</pre>
<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.	<pre>class Beer: def __init__(self): self.content = 1.0 def drink(self): self.content = 0.0 becks = Beer() # constructor - create class becks.drink() # beer empty: b.content == 0</pre>
<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).	<pre>x = int(input("your value: ")) if x > 3: print("Big") elif x == 3: print("Medium") else: print("Small")</pre>
<code>for, while</code>	<pre># For loop declaration for i in [0,1,2]: print(i)</pre>	<pre># While loop - same semantics j = 0 while j < 3: print(j) j = j + 1</pre>
<code>in</code>	Checks whether element is in sequence	<code>42 in [2, 39, 42] # True</code>
<code>is</code>	Checks whether both elements point to the same object	<pre>y = x = 3 x is y # True [3] is [3] # False</pre>
<code>None</code>	Empty value constant	<pre>def f(): x = 2 f() is None # True</pre>
<code>lambda</code>	Function with no name (anonymous function)	<code>(lambda x: x + 3)(3) # returns 6</code>
<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.	<pre>def incrementor(x): return x + 1 incrementor(4) # returns 5</pre>

Python Cheat Sheet: Functions and Tricks

“A puzzle a day to learn, code, and play” → Visit finxter.com

		Description	Example	Result
ADVANCED FUNCTIONALS	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')]
	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')]
TRICKS	python -m http.server <P>	Want to share files between PC and phone? Run this 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” → Visit finxter.com

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)<=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< L[0]]) + L[0:1] + qsort([x for x in L[1:] if x>=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)<=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>



Artificial General Intelligence (AGI): AGI refers to a hypothetical AI that can perform any intellectual task a human being can do, demonstrating human-like cognitive abilities across diverse domains.

Singularity: A theoretical point in the future when AI advancements lead to rapid, uncontrollable, and transformative changes in society, potentially surpassing human comprehension.

AI Safety: AI safety is the study and practice of building AI systems that operate securely and align with human values, ensuring that they benefit humanity without causing harm.

Alignment Problem: The alignment problem is the challenge of designing AI systems that understand and act upon human intentions, values, and goals, rather than optimizing for unintended objectives.

OpenAI: OpenAI is an AI research organization that focuses on developing artificial general intelligence (AGI) that benefits everybody.

Deep Learning: Deep learning is a subfield of machine learning that uses artificial neural networks to model complex patterns and make predictions or decisions based on input data.

Artificial Neural Network: An artificial neural network is a computational model inspired by the human brain's structure and function, consisting of interconnected nodes called neurons that process and transmit information.

Supervised Learning: Supervised learning is a machine learning approach where a model is trained on a dataset containing input-output pairs, learning to predict outputs based on new inputs.

Unsupervised Learning: Unsupervised learning is a machine learning approach where a model learns patterns and structures within input data without explicit output labels, often through clustering or dimensionality reduction.

Reinforcement Learning from Human Feedback (RLHF): RLHF is a method that combines reinforcement learning with human feedback, allowing AI models to learn from and adapt to human preferences and values.

Natural Language Processing (NLP): NLP is a field of AI that focuses on enabling computers to understand, interpret, and generate human language.

Large Language Models: Large language models are AI models trained on vast amounts of text data, capable of understanding and generating human-like text.

Transformer: The Transformer is a deep learning architecture designed for sequence-to-sequence tasks, known for its self-attention mechanism that helps capture long-range dependencies in data.

Attention mechanism: Attention mechanisms in neural networks enable models to weigh the importance of different input elements relative to one another, improving their ability to capture context.

Self-attention: Self-attention is a type of attention mechanism used in transformers that allows the model to relate different positions of a single sequence.

BERT (Bidirectional Encoder Representations from Transformers): BERT is a pre-trained transformer-based model developed by Google for natural language understanding tasks, which can be fine-tuned for specific applications.

GPT (Generative Pre-trained Transformer): GPT is a series of AI models developed by OpenAI, designed for natural language processing tasks and capable of generating coherent, contextually relevant text.

GPT-3.5: GPT-3.5 is an intermediate version of the GPT series, bridging the gap between GPT-3 and GPT-4 in terms of model size and capabilities.

GPT-4: GPT-4 is a more advanced version of the GPT series, expected to have larger model size and enhanced capabilities compared to its predecessors.

Pre-training: Pre-training is the initial phase of training a deep learning model on a large dataset, often unsupervised

Fine-tuning: Fine-tuning is the process of adapting a pre-trained model for a specific task by training it on labeled data related to that task, refining its performance.

Zero-shot learning: Zero-shot learning is a machine learning approach where a model can make predictions or complete tasks without being explicitly trained on that task's data.

Few-shot learning: Few-shot learning is a machine learning approach where a model can quickly adapt to new tasks by learning from a small number of labeled examples.

Token: A token is a unit of text, such as a word or subword, that serves as input to a language model.

Tokenizer: A tokenizer is a tool that breaks down text into individual tokens for processing by a language model.

Context window: The context window is the maximum number of tokens that a language model can process in a single pass, determining its ability to capture context in input data.

Prompts: Prompts are input text given to a language model to generate a response or complete a specific task.

Prompt Engineering: Prompt engineering is the process of designing effective prompts to elicit desired responses from language models, improving their utility and reliability.

ChatGPT: ChatGPT is a conversational AI model developed by OpenAI based on the GPT architecture, designed to generate human-like responses in text-based conversations.

InstructGPT: InstructGPT is an AI model developed by OpenAI, designed to follow instructions given in prompts, enabling it to generate more task-specific and accurate responses.

OpenAI API: The OpenAI API is a service provided by OpenAI that allows developers to access and utilize their AI models, such as ChatGPT, for various applications.

DALL-E: DALL-E is an AI model developed by OpenAI that generates images from textual descriptions, combining natural language understanding with image generation capabilities.

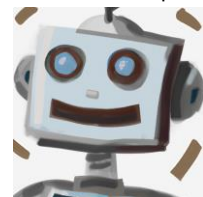
LaMDA: LaMDA is Google's conversational AI model designed to engage in open-domain conversations, understanding and generating responses for a wide range of topics.

Midjourney: AI program and service that generates images from natural language descriptions, called "prompts", similar to OpenAI's DALL-E and Stable Diffusion

Stable Diffusion: A deep learning, text-to-image model released in 2022 and used to generate detailed images conditioned on text descriptions. Also used for inpainting, outpainting, and generating image-to-image translations guided by a text prompt.

Diffusion models: Diffusion models are a class of models that represent the spread of information, influence, or other phenomena through a network.

Backpropagation: Backpropagation is a widely-used optimization algorithm in neural networks that minimizes the error between predicted outputs and true outputs by adjusting the model's weights.





Getting Started

Installation (CMD, Terminal, Shell, Powershell)

```
pip install openai
# or
pip3 install openai
```

First Prompt

```
import os
import openai

# Create, copy, and paste your API key here:
openai.api_key = "sk-123456789"

response = openai.Completion.create(
    model="text-davinci-003",
    prompt="2+2=",
    temperature=0, max_tokens=10)
```

Using GPT-4

```
system = 'You only reply in emojis!'
prompt = 'Who are you?'
```

```
res = openai.ChatCompletion.create(
    model="gpt-4",
    messages=[
        {"role": "system",
         "content": system},
        {"role": "user",
         "content": prompt}
    ],
    max_tokens=100,
    temperature=1.2)

print(res['choices'][0]['message']['content'])
# Answer:
```

JSON Output Format

```
{
  "choices": [
    {
      "finish_reason": "stop",
      "index": 0,
      "logprobs": null,
      "text": "4\n\n2+2=4"
    }
  ],
  "created": 1682409707,
  "id": "cmpl-797uNKSjEKE5cMlod1MeXkueIetkC",
  "model": "text-davinci-003",
  "object": "text_completion",
  "usage": {
    "completion_tokens": 8,
    "prompt_tokens": 4,
    "total_tokens": 12
  }
}
```

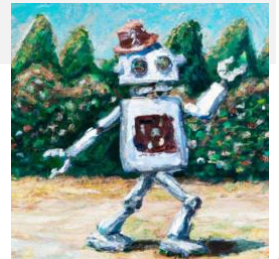
Generating Images Programmatically with DALL-E

```
prompt = "An oil painting of a dancing robot in the style of Monet"
```

```
response = openai.Image.create(
    prompt=prompt,
    n=1,
    size="256x256")
```

```
url = response["data"][0]["url"]
print(url)
# https://...
```

Resolution	Price
1024x1024	\$0.020 / image
512x512	\$0.018 / image
256x256	\$0.016 / image



Example Sentiment Analysis

```
prompt = """Do sentiment analysis on the following text. Text: 'Oh, I just adore how the sun shines so brightly at 5 a.m., waking me up every single morning!'"""

response = openai.Completion.create(
    engine="text-davinci-003",
    prompt=prompt,
    max_tokens=200,
    n=1,
    stop=None,
    temperature=0.5)

sentiment = response.choices[0].text.strip()
print(sentiment)
# Sentiment: Positive
```

Arguments Python OpenAI API Call

- ✓ **model**: Specifies the model version, e.g., 'gpt-4.0-turbo'.
- ✓ **prompt**: The input text for the model to process (e.g., question)
- ✓ **max_tokens**: Maximum tokens in the response. Roughly equates to number of words.
- ✓ **temperature**: Controls output randomness (0 to 1). Higher value leads to more random replies.
- ✓ **top_p**: Nucleus sampling strategy (0 to 1). Model will only consider subset of tokens whose probability exceeds top_p.
- ✓ **n**: Number of independent completions to explore.
- ✓ **stream**: Use streaming mode (True or False) to return results incrementally (e.g., for real-time apps).
- ✓ **echo**: Include input prompt in output (True or False).
- ✓ **stop**: Stopping sequence(s) for generation (string or list of strings).
- ✓ **presence_penalty**: Penalizes similar tokens in output.
- ✓ **frequency_penalty**: Penalizes frequent tokens in output.



Python Cheat Sheet: NumPy

“A puzzle a day to learn, code, and play” → Visit 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><slice> = <val></code>	Replace the <code><slice></code> as selected by the slicing operator with the value <code><val></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>