

נושא	שם הפונקציה / מתודה	אופן המימוש (Syntax)	הסבר קצר (בעברית)	איך כותבים את הפונקציה
בסיס וקלט/פלט	print()	print(value)	מדפיסה ערך או טקסט למסך הפלט.	אין
	input()	variable = input("text")	קולטת קלט מהמשתמש (תמיד כמחרוזת).	אין
	len()	len(sequence)	מחזירה את מספר האיברים ברצף או תווים במחרוזת.	<pre>def manual_len(iterable): count = 0 for _ in iterable: count += 1 return count</pre>
	type()	type(variable)	מחזירה את סוג הנתונים של המשתנה.	אין
	id()	id(obj)	מחזירה את הכתובת הייחודית של האובייקט בזיכרון.	אין
	help()	help(function)	מציגה הסבר ודוקומנטציה על פונקציה מסוימת.	אין
	dir()	dir(object)	מציגה את כל המתודות והתכונות של אובייקט.	אין
	int()	int(value)	ממירה ערך למספר שלם.	אין
המרת נתונים	float()	float(value)	ממירה ערך למספר עשרוני.	אין
	str()	str(value)	ממירה ערך למחרוזת (טקסט).	אין
	bool()	bool(value)	ממירה ערך ל-True או False.	אין
	list()	list(iterable)	הופכת רצף (כמו מחרוזת או טאפל) לרשימה.	אין
	dict()	dict(mapping)	יוצרת מילון חדש מתוך רצף של זוגות.	אין
	set()	set(iterable)	הופכת רצף לקבוצה (מוחקת כפילויות).	<pre>def manual_set(iterable=None): if iterable is None: return set() unique_items = [] for item in iterable: is_duplicate = False for existing in unique_items: if item == existing: is_duplicate = True break if not is_duplicate: unique_items.append(item) return {*unique_items}</pre>
	tuple()	tuple(iterable)	הופכת רצף לטאפל (רשימה שלא ניתנת לשינוי).	אין
	str.upper()	s.upper()	מחזירה עותק של המחרוזת באותיות גדולות.	<pre>def manual_upper(s): result = "" for char in s: if 'a' <= char <= 'z': result += chr(ord(char) - 32) else: result += char return result</pre>
	str.lower()	s.lower()	מחזירה עותק של המחרוזת באותיות קטנות.	<pre>def manual_lower(s): result = "" for char in s: if 'A' <= char <= 'Z': result += chr(ord(char) + 32) else: result += char return result</pre>

<pre>def manual_capitalize(s): if not s: return "" first = s[0] if 'a' <= first <= 'z': first = chr(ord(first) - 32) rest = manual_lower(s[1:]) return first + rest</pre>	הופכת רק את האות הראשונה במחרוזת לגדולה.	s.capitalize()	str.capitalize()	מחרוזות (Strings)
<pre>def manual_title(s): result = "" new_word = True for char in s: if char == " ": result += char new_word = True elif new_word: if 'a' <= char <= 'z': result += chr(ord(char) - 32) else: result += char new_word = False else: if 'A' <= char <= 'Z': result += chr(ord(char) + 32) else: result += char return result</pre>	הופכת את האות הראשונה בכל מילה לגדולה.	s.title()	str.title()	
<pre>def manual_strip(s): if not s: return "" start = 0 end = len(s) - 1 while start <= end and (s[start] == ' ' or s[start] == '\n' or s[start] == '\t'): start += 1 while end >= start and (s[end] == ' ' or s[end] == '\n' or s[end] == '\t'): end -= 1 return s[start:end+1]</pre>	מנקה רווחים (או תווים) מההתחלה ומהסוף.	s.strip()	str.strip()	
<pre>def manual_split(s, sep=' '): result = [] current_word = "" i = 0 sep_len = len(sep) while i < len(s): if s[i:i+sep_len] == sep: result.append(current_word) current_word = "" i += sep_len else: current_word += s[i] i += 1 result.append(current_word) return result</pre>	מפרקת מחרוזת לרשימה לפי תו מפריד.	s.split(sep)	str.split()	
<pre>def manual_join(iterable, sep): result = "" items = list(iterable) for i in range(len(items)): result += str(items[i]) if i < len(items) - 1: result += sep return result</pre>	מחברת רשימת מחרוזות למחרוזת אחת.	" ".join(list)	str.join()	

<pre>def manual_replace(s, old, new): if not old: return s result = "" i = 0 while i < len(s): if s[i : i + len(old)] == old: result += new i += len(old) else: result += s[i] i += 1 return result</pre>	מחליפה טקסט ישן בטקסט חדש.	s.replace(old, new)	str.replace()
<pre>def manual_find(s, sub): sub_len = len(sub) for i in range(len(s) - sub_len + 1): if s[i : i + sub_len] == sub: return i return -1</pre>	מחזירה אינדקס של מופע ראשון (או 1- אם לא נמצא).	s.find(sub)	str.find()
<pre>def manual_index(s, sub): result = manual_find(s, sub) if result == -1: raise ValueError("substring not found") return result</pre>	כמו find, אך זורקת שגיאה אם התת-מחרוזת לא נמצאה.	s.index(sub)	str.index()
<pre>def manual_isdigit(s): if not s: return False for char in s: if not ('0' <= char <= '9'): return False return True</pre>	בודקת האם המחרוזת מכילה רק ספרות.	s.isdigit()	str.isdigit()
<pre>def manual_isalpha(s): if not s: return False for char in s: if not (('a' <= char <= 'z' or 'A' <= char <= 'Z')): return False return True</pre>	בודקת האם המחרוזת מכילה רק אותיות.	s.isalpha()	str.isalpha()
<pre>def manual_startswith(s, prefix): if len(prefix) > len(s): return False for i in range(len(prefix)): if s[i] != prefix[i]: return False return True</pre>	בודקת האם המחרוזת מתחילה בטקסט מסוים.	s.startswith(prefix)	str.startswith()
<pre>def manual_endswith(s, suffix): if len(suffix) > len(s): return False start_point = len(s) - len(suffix) for i in range(len(suffix)): if s[start_point + i] != suffix[i]: return False return True</pre>	בודקת האם המחרוזת מסתיימת בטקסט מסוים.	s.endswith(suffix)	str.endswith()

<pre>def manual_count(s, sub): if not sub: return 0 count = 0 i = 0 while i <= len(s) - len(sub): if s[i : i + len(sub)] == sub: count += 1 i += len(sub) else: i += 1 return count</pre>	סופרת כמה פעמים מופיע תת-טקסט מסוים במחרוזת.	s.count(sub)	str.count()	
<pre>def manual_append(lst, item): lst[len(lst):] = [item]</pre>	מוסיפה איבר אחד לסוף הרשימה הקיימת.	l.append(x)	list.append()	רשימות (Lists)
<pre>def manual_insert(lst, index, item): if index < 0: index = max(0, len(lst) + index) if index > len(lst): index = len(lst) lst[index:index] = [item]</pre>	מוסיפה איבר במיקום i ספציפי ומזיזה את השאר.	l.insert(i, x)	list.insert()	
<pre>def manual_extend(lst, iterable): # מוסיפה איברים גם לעשות for item in iterable: lst[len(lst):] = [item]</pre>	מחברת את כל האיברים מרשימה אחרת לסוף הרשימה.	l.extend(iterable)	list.extend()	
<pre>def manual_pop(lst, index=-1): if not lst: raise IndexError("pop from empty list") if index == -1: index = len(lst) - 1 value = lst[index] lst[index : index + 1] = [] return value</pre>	מוחקת ומחזירה את האיבר במיקום ה-i (דיפולט: אחרון).	l.pop(i)	list.pop()	
<pre>def manual_remove(lst, value): found_index = -1 for i in range(len(lst)): if lst[i] == value: found_index = i break if found_index == -1: raise ValueError("list.remove(x): x not in list") lst[found_index : found_index + 1] = []</pre>	מוחקת את המופע הראשון של הערך x מהרשימה.	l.remove(x)	list.remove()	
<pre>def manual_sort(lst): n = len(lst) for i in range(n): for j in range(0, n - i - 1): if lst[j] > lst[j + 1]: lst[j], lst[j + 1] = lst[j + 1], lst[j]</pre>	ממיינת את הרשימה המקורית ("שינוי במקום").	l.sort()	list.sort()	
<pre>def manual_reverse(lst): n = len(lst) for i in range(n // 2): temp = lst[i] lst[i] = lst[n - 1 - i] lst[n - 1 - i] = temp</pre>	הופכת את סדר האיברים ברשימה המקורית ("במקום").	l.reverse()	list.reverse()	
<pre>def manual_index(lst, value): for i in range(len(lst)): if lst[i] == value: return i raise ValueError(f"{value} is not in list")</pre>	מחזירה את המיקום של הערך x הראשון שנמצא ברשימה.	l.index(x)	list.index()	
<pre>def manual_count(lst, value): count = 0 for item in lst: if item == value: count += 1 return count</pre>	סופרת כמה פעמים הערך x מופיע ברשימה.	l.count(x)	list.count()	
<pre>def manual_clear(lst): lst[:] = []</pre>	מוחקת את כל האיברים מהרשימה ומשאירה אותה ריקה.	l.clear()	list.clear()	

<pre>def manual_copy(lst): new_list = [] for item in lst: new_list[len(new_list):] = [item] return new_list</pre>	יוצרת עותק שטחי ונפרד של הרשימה.	<code>l.copy()</code>	<code>list.copy()</code>	
<pre>def manual_items(d): items_list = [] for key in d: items_list.append((key, d[key])) return items_list</pre>	מחזירה את כל זוגות המפתח-ערך כטאפלים (לולאות).	<code>d.items()</code>	<code>dict.items()</code>	מילונים (Dicts)
<pre>def manual_keys(d): keys_list = [] for key in d: keys_list.append(key) return keys_list</pre>	מחזירה רשימה (view) של כל המפתחות במילון.	<code>d.keys()</code>	<code>dict.keys()</code>	
<pre>def manual_values(d): values_list = [] for key in d: values_list.append(d[key]) return values_list</pre>	מחזירה רשימה (view) של כל הערכים במילון.	<code>d.values()</code>	<code>dict.values()</code>	
<pre>def manual_get(d, key, default=None): if key in d: return d[key] return default</pre>	שולפת ערך בבטחה (לא קורסת אם המפתח חסר).	<code>d.get(k, default)</code>	<code>dict.get()</code>	
<pre>def manual_pop(d, key, default=None): if key in d: value = d[key] del d[key] return value if default is not None: return default raise KeyError(key)</pre>	מוחקת מפתח ספציפי ומחזירה את הערך שלו.	<code>d.pop(key)</code>	<code>dict.pop()</code>	
<pre>def manual_popitem(d): if not d: raise KeyError("popitem(): dictionary is empty") last_key = list(d)[-1] value = d[last_key] del d[last_key] return (last_key, value)</pre>	מוחקת ומחזירה את הזוג האחרון שנכנס (מפתח, ערך).	<code>d.popitem()</code>	<code>dict.popitem()</code>	
<pre>def manual_update(d, other_dict): for key in other_dict: d[key] = other_dict[key]</pre>	מעדכנת את המילון עם נתונים ממילון אחר או רצף זוגות.	<code>d.update(other)</code>	<code>dict.update()</code>	
<pre>def manual_copy(d): new_dict = {} for key in d: new_dict[key] = d[key] return new_dict</pre>	יוצרת עותק רדוד של המילון.	<code>d.copy()</code>	<code>dict.copy()</code>	
<pre>def manual_clear(d): keys = list(d) for key in keys: del d[key]</pre>	מוחקת את כל התוכן של המילון.	<code>d.clear()</code>	<code>dict.clear()</code>	
<pre>def manual_abs(x): if x < 0: return -x return x</pre>	מחזירה ערך מוחלט (מבטלת סימן מינוס).	<code>abs(number)</code>	<code>abs()</code>	
<pre>def manual_round(number, ndigits=0): multiplier = 10 ** ndigits shifted = number * multiplier if shifted >= 0: rounded = int(shifted + 0.5) else: rounded = int(shifted - 0.5) return rounded / multiplier if ndigits > 0 else int(rounded / multiplier)</pre>	מעגלת מספר לפי כמות ספרות רצויה אחרי הנקודה.	<code>round(n, digits)</code>	<code>round()</code>	
<pre>def manual_sum(iterable, start=0): total = start for item in iterable: total += item return total</pre>	מחשבת סכום של כל המספרים ברצף.	<code>sum(iterable)</code>	<code>sum()</code>	

<pre>def manual_max(iterable): it = iter(iterable) try: current_max = next(it) except StopIteration: raise ValueError("max() arg is an empty sequence") for item in it: if item > current_max: current_max = item return current_max</pre>	מחזירה את האיבר הגדול ביותר ברצף.	max(iterable)	max()
<pre>def manual_min(iterable): it = iter(iterable) try: current_min = next(it) except StopIteration: raise ValueError("min() arg is an empty sequence") for item in it: if item < current_min: current_min = item return current_min</pre>	מחזירה את האיבר הקטן ביותר ברצף.	min(iterable)	min()
<pre>def manual_range(start, stop=None, step=1): if stop is None: stop = start start = 0 if step == 0: raise ValueError("range() arg 3 must not be zero") current = start if step > 0: while current < stop: yield current current += step else: while current > stop: yield current current += step</pre>	מייצרת רצף מספרים מהתחלה ועד אחד לפני הסוף.	range(start, stop)	range()
<pre>def manual_enumerate(iterable, start=0): n = start for elem in iterable: yield n, elem n += 1</pre>	נותנת אינדקס וערך בו-זמנית בתוך לולאת for.	enumerate(seq)	enumerate()
<pre>def manual_zip(*iterables): iterators = [iter(it) for it in iterables] while True: try: current_tuple = [] for it in iterators: current_tuple.append(next(it)) yield tuple(current_tuple) except StopIteration: break</pre>	מחברת איברים מרשימות שונות לזוגות לפי הסדר.	zip(l1, l2)	zip()

מתמטיקה ואיטרציה

<pre>def manual_sorted(iterable, reverse=False): new_list = [] for item in iterable: new_list.append(item) n = len(new_list) for i in range(n): for j in range(0, n - i - 1): if not reverse: if new_list[j] > new_list[j + 1]: new_list[j], new_list[j + 1] = new_list[j + 1], new_list[j] else: if new_list[j] < new_list[j + 1]: new_list[j], new_list[j + 1] = new_list[j + 1], new_list[j] return new_list</pre>	מחזירה רשימה חדשה וממוינת (לא משנה את המקור).	sorted(iterable)	sorted()	
<pre>def manual_reversed(seq): for i in range(len(seq) - 1, -1, -1): yield seq[i]</pre>	מחזירה איטרטור שהופך את סדר האיברים (ללא שינוי מקור).	reversed(seq)	reversed()	
<pre>def manual_any(iterable): for item in iterable: if item: return True return False</pre>	מחזירה True אם לפחות איבר אחד ברצף הוא "אמת".	any(iterable)	any()	
<pre>def manual_all(iterable): for item in iterable: if not item: return False return True</pre>	מחזירה True אם כל האיברים ברצף הם "אמת".	all(iterable)	all()	
<pre>def manual_add(s, element): exists = False for item in s: if item == element: exists = True break if not exists: s = {element}</pre>	מוסיפה איבר ייחודי לקבוצה.	s.add(x)	set.add()	סטים וקבצים
<pre>def manual_remove(s, element): found = False for item in s: if item == element: found = True break if found: s -= {element} else: raise KeyError(element)</pre>	מוחקת איבר (זורקת שגיאה אם האיבר לא קיים).	s.remove(x)	set.remove()	
<pre>def manual_discard(s, element): found = False for item in s: if item == element: found = True break if found: s -= {element}</pre>	מוחקת איבר (לא זורקת שגיאה אם האיבר חסר).	s.discard(x)	set.discard()	
ל.ר.	פותחת קובץ לקריאה ('r'), כתיבה ('w') או הוספה ('a').	open(file, mode)	open()	