

Python Exam Cheatsheet — 20606

Anastasia Zarankin & Yehonatan Simian

1 Instructions

1.1 Staff Letter

- All code must be well documented.
- Begin algorithms with a brief explanation.
- Algorithms must be as efficient as possible.
- Allowed functions are described in section 1.3.

1.2 Forum Instructions

- Helper functions may be defined nestedly.
- A Sliced list is not considered as a new list.
- `in` is considered $O(1)$ (constant time).
- `min`, `max` on a list are considered $O(n)$.

1.3 Allowed Functions

1.3.1 Built-in Functions

- | | | | |
|----------------------|---------------------------|----------------------|-----------------------|
| • <code>abs</code> | • <code>isinstance</code> | • <code>min</code> | • <code>sorted</code> |
| • <code>float</code> | • <code>len</code> | • <code>pow</code> | • <code>str</code> |
| • <code>input</code> | • <code>list</code> | • <code>print</code> | • <code>sum</code> |
| • <code>int</code> | • <code>max</code> | • <code>range</code> | • <code>tuple</code> |

```
1 abs(-5)           # = 5
2 float(5)          # = 5.0
3 int("5")           # = 5
4 isinstance(5, int) # = True
5 len("Hello")       # = 5
6 list("Bye")        # = ['B', 'y', 'e']
7 max(1, 5, 3)       # = 5

1 min(1, 5, 3)       # = 1
2 pow(2, 3)          # = 8
3 list(range(5))     # = [0, 1, 2, 3, 4]
4 sorted([3, 1, 2])  # = [1, 2, 3]
5 str(5)             # = '5'
6 sum([1, 2, 3])     # = 6
7 tuple([1, 2, 3])   # = (1, 2, 3)

1 user_input = input("Enter something: ") # Waits for user input (String)
```

1.3.2 String Methods

- | | | |
|-----------|-------------------|------------------|
| • slicing | • <code>in</code> | • <code>+</code> |
|-----------|-------------------|------------------|

```
1 s = "Hello"
2 print(s[1:-1:2])    # Output: 'el'
3 print("Hell" in s)  # Output: True
4 print(s + ", Goodbye") # Output: 'Hello, Goodbye'
```

1.3.3 List Methods

- | | | | |
|-------------------|---------------------|---------------------|-----------------------|
| • slicing | • <code>+</code> | • <code>pop</code> | • <code>append</code> |
| • <code>in</code> | • <code>sort</code> | • <code>copy</code> | |

```
1 l1 = [1, 2.0, "3"]
2 l2 = [9, 2.0]
3 print(l1[1:-1:2])  # Output: [2.0]
4 print(2 in l1)     # Output: True
5 print(l1 + l2)     # Output: [1, 2.0, '3', 9, 2.0]
6 l2.sort()          # l2 is now [2.0, 9]
7 l1.sort()          # TypeError: '<' not supported...
8 l1.pop()           # l1 is now [1, 2.0]
9 l1.sort()          # l1 is now [1, 2.0]
10 l3 = l1.copy()     # l3 is now [1, 2.0]
11 l1.append('hi')    # l1 is now [1, 2.0, 'hi']
```

2 Exceptions

```
1 def handle_exception():
2     try:
3         # Code that may raise an exception
4     except ExceptionType as e:
5         # Handle the exception, e.g., print(e)
6     except (ExceptionType1, ExceptionType2) as e:
7         # Handle multiple exceptions
8     except:
9         # Handle all other exceptions
10    else:
11        # Code that runs if no exception was raised
12    finally:
13        # Code that always runs
```

3 Useful Functions

```
1 def index_of(num, lst):
2     if not lst:
3         return -1
4     if lst[0] == num:
5         return 0
6     index = index_of(num, lst[1:])
7     if index != -1:
8         index += 1
9     return index
```

```
1 def is_prime(n, i=2):
2     if n <= 2:
3         return n == 2
4     if n % i == 0:
5         return False
6     if i * i > n:
7         return True
8
9     return is_prime(n, i + 1)
```

```
1 def is_sorted(lst):
2     if len(lst) <= 1:
3         return True
4     if lst[0] > lst[1]:
5         return False
6     return is_sorted(lst[1:])
```

```
1 def exist(num, lst):
2     if not lst:
3         return False
4     if lst[0] == num:
5         return True
6     return exist(num, lst[1:])
```

```
1 def is_palindrome(s):
2     if len(s) <= 1:
3         return True
4     if s[0] != s[-1]:
5         return False
6     return is_palindrome(s[1:-1])
```

```
1 def is_square(mat):
2     if len(mat) == 0:
3         return False
4     for row in mat:
5         if len(row) != len(mat):
6             return False
7     return True
```

```
1 def is_power_of_2(n):
2     if n == 1:
3         return True
4     if n % 2 != 0 or n == 0:
5         return False
6     return is_power_of_2(n // 2)
```

```
1 def bubble_sort(lst):
2     for i in range(len(lst)):
3         for j in range(len(lst) - 1):
4             if lst[j] > lst[j+1]:
5                 lst[j], lst[j+1] = lst[j+1], lst[j]
6     return lst
```

```
1 def max_sort(lst):
2     if len(lst) == 1:
3         return lst
4     max_index = lst.index(max(lst))
5     lst[max_index], lst[-1] = lst[-1], lst[max_index]
6     return max_sort(lst[:-1]) + [lst[-1]]
```

4 Preparation Exercises

```
1 def maximal_drop(lst):
2     # Calculate the maximal drop between two heights in a list.
3     if not lst:
4         return 0
5
6     max_drop = 0
7     max_height_so_far = lst[0]
8
9     for height in lst:
10        if height > max_height_so_far:
11            max_height_so_far = height
12        else:
13            drop = max_height_so_far - height
14            if drop > max_drop:
15                max_drop = drop
16
17    return max_drop
```

```
1 def is_serpertine(mat):
2     if not is_square(mat) or mat[0][0] != 1:
3         return False
4
5     n = len(mat)
6     for i in range(n):
7         for j in range(1, n):
8             if i % 2 == 0 and not mat[i][j] - mat[i][j - 1] == 1:
9                 return False # Check ascending order for even rows
10            elif i % 2 == 1 and not mat[i][j - 1] - mat[i][j] == 1:
11                return False # Check descending order for odd rows
12
13    return True
```

```
1 def is_identity(mat, x, size):
2     if not is_square(mat) or x < 0 or x + size > len(mat) or size < 1:
3         return False
4
5     for i in range(size):
6         for j in range(size):
7             if (i == j and mat[x + i][x + j] != 1) or (
8                 i != j and mat[x + i][x + j] != 0
9             ):
10                return False
11
12    return True
```

```
1 def bulls_and_cows(number, guess):
2     def helper(number, guess, guess_index):
3         if guess_index >= len(guess):
4             return 0
5         number_index = index_of(guess[guess_index], number)
6         points = 0
7         if number_index != -1:
8             points += 1
9         if number_index == guess_index:
10            points += 1
11        return points + helper(number, guess, guess_index + 1)
12
13    return helper(number, guess, 0)
```

```

1 def find_pair(sum, lst):
2     # Check if there are two numbers in the
3     # list that sum up to 'sum'.
4     if not lst:
5         return False
6     if exist(sum - lst[0], lst[1:]):
7         return True
8     return find_pair(sum, lst[1:])

```

```

1 def max_matrix(mat):
2     # Maximum size of an identity central
3     # submatrix of square and odd sized 'mat'.
4     n = len(mat)
5     for x in range(n // 2 + 1):
6         size = n - x * 2
7         if is_identity(mat, x, size):
8             return size
9     return 0

```

```

1 def minus_plus(lst):
2     # Check if each num has a negative twin.
3     if len(lst) % 2 != 0:
4         return False
5
6     def helper(sublist):
7         if not sublist:
8             return True
9         twin = exist(-sublist[0], lst)
10        return twin and helper(sublist[1:])
11
12    return helper(lst)

```

```

1 def secret(s1, s2, key):
2     # Check if s2 is derived from the s1.
3     if len(s1) != len(s2):
4         return False
5
6     def helper(index):
7         if index == len(s1):
8             return True
9         c = ord(s1[index]) + key + index
10        if c != ord(s2[index]):
11            return False
12        return helper(index + 1)
13    return helper(0)

```

```

1 def max_mul2(lst):
2     # Find the largest possible product of two
3     # elements in a list.
4     max1 = max2 = float("-inf")
5     min1 = min2 = float("inf")
6     for num in lst:
7         # Update the two largest values
8         if num > max1:
9             max2 = max1
10            max1 = num
11        elif num > max2:
12            max2 = num
13        # Update the two smallest values
14        if num < min1:
15            min2 = min1
16            min1 = num
17        elif num < min2:
18            min2 = num
19    return max(max1 * max2, min1 * min2)

```

```

1 def print_pairs(arr, k):
2     # Print all pairs in the list whose
3     # difference is exactly k.
4     n = len(arr)
5     if n < 2:
6         return
7     left, right = 0, 1
8     while right < n:
9         diff = arr[right] - arr[left]
10        if diff == k:
11            print(f"({arr[left]}, {arr[right]})")
12            left += 1
13            right += 1
14        elif diff > k:
15            left += 1
16        if left == right:
17            right += 1
18    else:
19        right += 1

```

4.1 Exercise 10: Coffee Shop

```

1 class Date:
2     def __init__(self, d, m, y):
3         self._day = d
4         self._month = m
5         self._year = y
6
7     def __eq__(self, other):
8         return (isinstance(other, Date)
9                 and self._year == other._year
10                and self._month == other._month
11                and self._day == other._day)
12
13    def __lt__(self, other):
14        if not isinstance(other, Date): return False # shortening a bit space
15        if self._year < other._year: return True
16        if self._year > other._year: return False
17        if self._month < other._month: return True
18        if self._month > other._month: return False
19        return self._day < other._day

```

```

1 class Order:
2     _order_num = 1
3
4     def __init__(self, day, month, year, hour, minute, cost=50):
5         self._t = Time(hour, minute)
6         self._d = Date(day, month, year)
7         self._cost = cost
8         self._order_id = Order._order_num
9         Order._order_num += 1
10
11     def __gt__(self, other):
12         return isinstance(other, Order) and self._cost > other._cost

```

```

1 class CashRegister:
2     def __init__(self):
3         self._orders = []
4
5     def add_order(self, order):
6         self._orders.append(order)
7
8     def monthly_total_income(self, month):
9         return sum([order._cost for order in self._orders if order._d._month == month])
10
11     def most_expensive_order(self, date):
12         return max([order for order in self._orders if order._d == date])._order_id
13
14     def less_than(self, cost):
15         filtered_orders = [order for order in self._orders if order._cost < cost]
16         return filtered_orders if filtered_orders else None

```

4.2 Exercise 11: Contacts List

```

1 class Person:
2     def __init__(self, name, id, birth):
3         self._name = name
4         self._id = id
5         self._birth = birth
6
7     def __eq__(self, other):
8         return isinstance(other, Person) and self._id == other._id

```

```

1 class ContactsList:
2     def __init__(self) -> None:
3         self._contacts = []
4
5     def born_in_date(self, d):
6         return [contact for contact in self._contacts if contact._birth == d]
7
8     def oldest_contact(self):
9         def get_birth(contact):
10             return contact._birth
11
12         return min(self._contacts, key=get_birth)
13
14     def born_in_month(self):
15         months = [0] * 13
16         for contact in self._contacts:
17             months[contact._birth._month] += 1
18         return [(i, months[i]) for i in range(1, 13)]

```

5 Maman 13

Note:

- Maman 11 is `bulls_and_cows` that's implemented in section 4.
- Maman 12 is `tic_tac_toe` and I highly doubt that such thing will be on the exam.
- Maman 14 is just classes, nothing that's not covered on section 4.1 and section 4.2.

```

1 def find_missing_item_linear(lst):
2     # Find the only missing number in an arithmetic sequence of at least four numbers
3     diffs = []
4     for i in range(3):
5         diff = lst[i + 1] - lst[i]
6         if diff in (diffs):
7             correct_diff = diff
8             break
9     diffs.append(diff)
10
11     correct_diff = (lst[-1] - lst[0]) // len(lst)
12     for i in range(len(lst) - 1):
13         if lst[i + 1] - lst[i] != correct_diff:
14             return lst[i] + correct_diff

```

```

1 def find_missing_item_logarithmic(lst): # Same as above but logarithmic.
2     diffs = ... # same as lines 3-9 in the previous function
3
4     left, right = 0, len(lst) - 1
5     while left <= right: # Find the missing number using binary search
6         mid = (left + right) // 2
7         expected = lst[0] + mid * correct_diff # a_n=a_1+(n-1)*d
8         if lst[mid] != expected:
9             if lst[mid - 1] == lst[0] + (mid - 1) * correct_diff:
10                 return expected
11             right = mid - 1
12         else:
13             left = mid + 1

```

```

1 def split_list_index(lst):
2     # Find the index to split a list into two parts with equal sums.
3     if len(lst) < 2:
4         return -1
5     total_sum, left_sum = sum(lst), 0
6     for i in range(len(lst)):
7         left_sum += lst[i]
8         total_sum -= lst[i]
9         if left_sum == total_sum:
10             return i
11     return -1

```

```

1 def max_sequence(lst, prev_last_digit=None, current_length=0, max_length=0):
2     # Return the length of the longest sequence of numbers with the same first and last
3     # digits. Assume that 'lst' is not empty and contains only integers.
4     if not lst:
5         return max_length
6     first, *rest = lst
7     first_last_digit, first_first_digit = first % 10, int(str(first)[0])
8     if prev_last_digit is None or prev_last_digit == first_first_digit:
9         current_length += 1
10        max_length = max(max_length, current_length)
11    else:
12        current_length = 1
13    return max_sequence(rest, first_last_digit, current_length, max_length)

```

```

1 def order(str1, str2):
2     # Merge two ordered strings into one ordered string. Assume that the strings contain
3     # only lowercase letters, are ordered in ascending order, and may have different lengths.
4     if not str1:
5         return str2
6     if not str2:
7         return str1
8     if str1[0] < str2[0]:
9         return str1[0] + order(str1[1:], str2)
10    return str2[0] + order(str1, str2[1:])

```

6 Exam 26.06.2020

6.1 Question 1

```
1 def biggest_sum(lst):
2     # Return the biggest sum of integers between 2 zeroes in 'lst'.
3     max_sum, curr_sum = 0, 0
4
5     for num in lst[index_of(0, lst) + 1 :]:
6         if num == 0:
7             max_sum = max(max_sum, curr_sum)
8             curr_sum = 0
9         else:
10            curr_sum += num
11
12    return max_sum
```

```
1 def biggest_sum_row(mat):
2     # Return the index of the row with the biggest_sum.
3     if not mat:
4         return None
5
6     max_sum, max_index = 0, 0
7
8     for i in range(len(mat)):
9         curr_sum = biggest_sum(mat[i])
10        if curr_sum > max_sum:
11            max_sum = curr_sum
12            max_index = i
13
14    return max_index
```

6.2 Question 2

```
1 def common(lst1, lst2):
2     # Return the common elements of both lists.
3     commons_lst = []
4     i, j = 0, 0
5     while i < len(lst1) and j < len(lst2):
6         if lst1[i] == lst2[j]:
7             commons_lst.append(lst1[i])
8             i += 1
9             j += 1
10        elif lst1[i] > lst2[j]:
11            j += 1
12        else:
13            i += 1
14
15    return commons_lst or None
```

6.3 Question 3

```
1 def max_pos_seq(lst):
2     # Return the number of highest sequence of consecutive positive numbers in 'lst'.
3
4     def helper(lst, curr_seq):
5         if not lst:
6             return curr_seq
7         if lst[0] <= 0:
8             return max(curr_seq, helper(lst[1:], 0))
9         return helper(lst[1:], curr_seq + 1)
10
11    return helper(lst, 0)
```

```

1 def is_palindrome(lst):
2     # Return whether all elements, as well as the list itself, are palindromes.
3     if not lst:
4         return True
5
6     begin, end = lst[0], lst[-1]
7
8     return begin == end == begin[::-1] and is_palindrome(lst[1:-1])

```

6.4 Question 4

Question 4 is boring and nothing that's not covered on section 4.1 and section 4.2.

6.5 Question 5

```

1 def mirror_list(mat):
2     # Check if the matrix is a mirror list with minimum comparisons.
3     # Note: I'm not sure if this is the best solution.
4     if not mat or not mat[0]:
5         return False
6
7     def check_equality_2(e1, e2):
8         if not isinstance(e1, str) or not isinstance(e2, str):
9             raise TypeError("The matrix must contain only strings")
10        if len(e1) != 1 or len(e2) != 1:
11            raise ValueError("The matrix must contain only single characters")
12        return e1 == e2
13
14    def check_equality_4(e1, e2, e3, e4):
15        return check_equality_2(e1, e2) and check_equality_2(e3, e4) and e2 == e3
16
17    mid_x = len(mat) // 2
18    mid_y = len(mat[0]) // 2
19    for i in range(mid_x):
20        for j in range(mid_y):
21            if not check_equality_4(mat[i][j], mat[i][~j], mat[~i][j], mat[~i][~j]):
22                return False
23
24    if len(mat) % 2 == 1:
25        for i in range(mid_x + 1):
26            if not check_equality_2(mat[mid_x][i], mat[mid_x][~i]):
27                return False
28
29    if len(mat[0]) % 2 == 1:
30        for j in range(mid_y + 1):
31            if not check_equality_2(mat[j][mid_y], mat[~j][mid_y]):
32                return False
33
34    return True

```

7 Yo Mama Jokes

I have some space left, so here are some Yo Mama jokes generated by GitHub Copilot:

- Yo mama is so fat, when she wears a yellow raincoat, people yell, "Taxi!"
- Yo mama is so stupid, she put two quarters in her ears and thought she was listening to 50 Cent.
- Yo mama is so old, when she was in school, history was called current events.
- Yo mama is so short, you can see her feet on her driver's license.
- Yo mama is so ugly, when she walks into a bank, they turn off the cameras.
- Yo mama is so poor, she walked down the road with one shoe. And if you ask her if she lost a shoe, she'd say "No, I found one."

Thank you.