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 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

- abs
- float
- input
- int
- isinstance
- len
- list
- max
- min
- pow
- print
- range
- sorted
- str
- sum
- tuple

```python
abs(-5)            # = 5
float(5)           # = 5.0
int("5")           # = 5
isinstance(5, int) # = True
len("Hello")       # = 5
list("Bye")        # = [’B’, ’y’, ’e’]
max(1, 5, 3)       # = 5
```

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

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

1.3.2 String Methods

- slicing
- in
- +

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

1.3.3 List Methods

- slicing
- in
- +
- sort
- pop
- copy
- append

```python
  l1 = [1, 2.0, "3"]
  l2 = [9, 2.0]
  print(l1[1:-1:2])  # Output: [2.0]
  print(2 in l1)     # Output: True
  print(l1 + l2)     # Output: [1, 2.0, ’3’, 9, 2.0]
  l2.sort()          # l2 is now [2.0, 9]
  l1.sort()          # TypeError: ’<’ not supported...
  l1.pop()           # l1 is now [1, 2.0]
  l1.sort()          # l1 is now [1, 2.0]
  l3 = l1.copy()     # l3 is now [1, 2.0]
  l1.append(’hi’)    # l1 is now [1, 2.0, ’hi’]
```

2 Useful Functions

```python
def index_of(num, lst):
  if not lst:
    return -1
  if lst[0] == num:
    return 0
  index = index_of(num, lst[1:])
  if index != -1:
    index += 1
  return index
```

```python
def is_sorted(lst):
  if len(lst) <= 1:
    return True
  if lst[0] > lst[1]:
    return False
  return is_sorted(lst[1:])
```

```python
def is_palindrome(s):
  if len(s) <= 1:
    return True
  if s[0] != s[-1]:
    return False
  return is_palindrome(s[1:-1])
```

```python
def is_power_of_2(n):
  if n == 1:
    return True
  if n % 2 != 0 or n == 0:
    return False
  return is_power_of_2(n // 2)
```

```python
def is_prime(n, i=2):
  if n <= 2:
    return n == 2
  if n % i == 0:
    return False
  if i * i > n:
    return True

  return is_prime(n, i + 1)
```

```python
def exist(num, lst):
  if not lst:
      return False
  if lst[0] == num:
      return True
  return exist(num, lst[1:])
```

```python
def is_square(mat):
  if len(mat) == 0:
      return False
  for row in mat:
      if len(row) != len(mat):
          return False
  return True
```

```python
def bubble_sort(lst):
  for i in range(len(lst)):
    for j in range(len(lst) - 1):
      if lst[j] > lst[j+1]:
        lst[j], lst[j+1] = lst[j+1], lst[j]
  return lst
```

```python
def max_sort(lst):
  if len(lst) == 1:
    return lst
  max_index = lst.index(max(lst))
  lst[max_index], lst[-1] = lst[-1], lst[max_index]
  return max_sort(lst[:-1]) + [lst[-1]]
```

3 Preparation Exercises

```python
def is_serpertine(mat):
  if not is_square(mat) or mat[0][0] != 1:
      return False

  n = len(mat)
  for i in range(n):
    for j in range(1, n):
      if i % 2 == 0 and not mat[i][j] - mat[i][j - 1] == 1:
        return False # Check ascending order for even rows
      elif i % 2 == 1 and not mat[i][j - 1] - mat[i][j] == 1:
        return False # Check descending order for odd rows

  return True
```

```python
def is_identity(mat, x, size):
  if not is_square(mat) or x < 0 or x + size > len(mat) or size < 1:
    return False

  for i in range(size):
    for j in range(size):
      if (i == j and mat[x + i][x + j] != 1) or (
          i != j and mat[x + i][x + j] != 0
      ):
        return False

  return True
```

```
def find_pair(sum, lst):
  # Check if there are two numbers in the list that sum up to ‘sum‘.
  if not lst:
    return False
  if exist(sum - lst[0], lst[1:]):
    return True

  return find_pair(sum, lst[1:])
```

```
def max_matrix(mat):
  # Maximum size of an identity central submatrix of square and odd sized ‘mat‘.
  n = len(mat)
  for x in range(n // 2 + 1):
    size = n - x * 2
    if is_identity(mat, x, size):
      return size
  return 0
```

```
def minus_plus(lst):
  # Check if each element has a negative twin.
  if len(lst) % 2 != 0:
    return False

  def helper(sublist):
    if not sublist:
      return True
  return exist(-sublist[0], lst) and helper(sublist[1:])

  return helper(lst)
```

```
def max_mul2(lst):
  # Find the largest possible product of two elements in a list.
  max1 = max2 = float("-inf")
  min1 = min2 = float("inf")

  for num in lst:
    # Update the two largest values
    if num > max1:
      max2 = max1
      max1 = num
    elif num > max2:
      max2 = num

    # Update the two smallest values
    if num < min1:
      min2 = min1
      min1 = num
    elif num < min2:
      min2 = num

  return max(max1 * max2, min1 * min2)
```

```
def secret(s1, s2, key):
  # Determines if the second string is derived from the first string using the key.
  if len(s1) != len(s2):
    return False

  def helper(index):
    if index == len(s1):
      return True
```

```python
    if ord(s1[index]) + key + index != ord(s2[index]):
      return False
    return helper(index + 1)

  return helper(0)
```

```python
# TODO: Implement
```

```python
def print_pairs(arr, k):
  # Print all pairs in the list whose difference is exactly k.
  n = len(arr)
  if n < 2:
    return
  left, right = 0, 1
  while right < n:
    diff = arr[right] - arr[left]
    if diff == k:
      print(f"({arr[left]}, {arr[right]})")
      left += 1
      right += 1
    elif diff > k:
      left += 1
      if left == right:
        right += 1
    else:
      right += 1
```

```python
def maximal_drop(lst):
  # Calculate the maximal drop between two heights in a list.
  if not lst:
    return 0

  max_drop = 0
  max_height_so_far = lst[0]

  for height in lst:
    if height > max_height_so_far:
      max_height_so_far = height
    else:
      drop = max_height_so_far - height
      if drop > max_drop:
        max_drop = drop

  return max_drop
```

3.1 Exercise 10: Coffee Shop

```python
class Date:
  def __init__(self, d, m, y):
    self._day = d
    self._month = m
    self._year = y

  def __eq__(self, other):
    return (
      isinstance(other, Date)
      and self._year == other._year
      and self._month == other._month
      and self._day == other._day
    )

  def __lt__(self, other):
    if not isinstance(other, Date):
      return False
    if self._year < other._year:
      return True
    if self._year > other._year:
      return False
    if self._month < other._month:
      return True
    if self._month > other._month:
      return False
    return self._day < other._day
```

```
class Order:
  _order_num = 1

  def __init__(self, day, month, year, hour, minute, cost=50):
    self._t = Time(hour, minute)
    self._d = Date(day, month, year)
    self._cost = cost
    self._order_id = Order._order_num
    Order._order_num += 1

  def __gt__(self, other):
    return isinstance(other, Order) and self._cost > other._cost
```

```
class CashRegister:
  def __init__(self):
    self._orders = []

  def add_order(self, order):
    self._orders.append(order)

  def monthly_total_income(self, month):
    return sum([order._cost for order in self._orders if order._d._month == month])

  def most_expensive_order(self, date):
    return max([order for order in self._orders if order._d == date])._order_id

  def less_than(self, cost):
    filtered_orders = [order for order in self._orders if order._cost < cost]
    return filtered_orders if filtered_orders else None
```

3.2 Exercise 11: Contacts List

```
class Person:
  def __init__(self, name, id, birth):
    self._name = name
    self._id = id
    self._birth = birth

  def __eq__(self, other):
    return isinstance(other, Person) and self._id == other._id
```

```
class ContactsList:
  def __init__(self) -> None:
    self._contacts = []

  def born_in_date(self, d):
    return [contact for contact in self._contacts if contact._birth == d]

  def oldest_contact(self):
    def get_birth(contact):
      return contact._birth

    return min(self._contacts, key=get_birth)

  def born_in_month(self):
    months = [0] * 13
    for contact in self._contacts:
      months[contact._birth._month] += 1
    return [(i, months[i]) for i in range(1, 13)]
```