Under the Hood of the Snake

Particle and Nuclear Physics PhD Day

Sten Åstrand, December 11th, 2025

Previously...

PHYSICS IN PILEUP

Hidden in plain sight

PhD Day, June 2024

Sten Åstrand

8

[Inverted CERN School of Computing 2025, video recording]

CERN School of Computing

46th CERN School of Computing

LUND UNIVERSITY

6-19 July
Lund, Sweden

CERN

“Python is slow!”

python™

Under the Hood of the Snake

Behind the scenes of Python

Sten Åstrand
March 25th
Inverted CERN School of Computing 2025

"Python is slow!"

What does it mean?

Why is this?

What do we do?

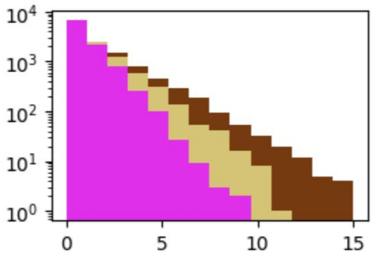
Jupyter Notebook

[45]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[46]:
```
def random_points(n_points, decay, random_seed):
    rng = np.random.default_rng(seed=random_seed)
    points = rng.exponential(scale=decay, size=n_points)
    return points

def random_color(random_seed):
    rng = np.random.default_rng(seed=random_seed)
    return rng.random(size=3)
```

[47]:
```
n_curves = 3
fig, ax = plt.subplots(figsize=(3,2))
bins = np.linspace(0, 15, 15)
for curve_idx in range(n_curves-1, -1, -1):
    points = random_points(10000, 1 + curve_idx/2, 42 + 4*curve_idx)
    ax.hist(
        points,
        bins=bins,
        color=random_color(1337 + 8*curve_idx),
        linewidth=2
    )
plt.yscale("log")
```



Issues

- specific, disparate solutions
- binary results
- success without reflection

Issues		Goals
specific, disparate solutions	➔	considering options
binary results	➔	results with different merits
success without reflection	➔	success through iteration, reflection

The exercises

Count instances

```python
def solution(lst, number):
    count = 0
    for element in lst:
        if element == number:
            count += 1
    return count
```

Normalize values

```python
def normalize(value, normalize_by):
    return value / normalize_by

def solution(lst):
    new_lst = []
    for value in lst:
        new_value = normalize(value, max(lst))
        new_lst.append(new_value)
    return new_lst
```

Clamp matrix values

```python
def solution(matrix):
    # Iterate over rows
    for i in range(len(matrix)):
        # Iterate over columns
        for j in range(len(matrix[i])):
            if matrix[i][j] < 0:
                matrix[i][j] = 0
            elif matrix[i][j] > 255:
                matrix[i][j] = 255
    return matrix
```

n largest numbers

```python
def solution(lst, n):
    largest = []
    for i in (range(n)):
        this_largest = 0
        largest_index = 0
        for n, num in enumerate(lst):
            # Check if the current number is the largest so far,
            # and store it and its index
            if num > this_largest:
                this_largest = num
                largest_index = n
        # Store the largest number found, and remove it from lst
        largest.append(this_largest)
        lst.pop(largest_index)
    return largest
```

Data processing

```python
def solution(file):
    data_table = []
    with open(file, "r") as fp:
        # Read in data from csv row by row
        reader = csv.reader(fp)
        for row in reader:
            data_table.append(row)

    headers = data_table[0]
    masses = []
    for row in data_table[1:]:
        if row[1] == "signal":
            # The current row is classified as 'signal'
            masses.append(float(row[5]))

    mass_mean = sum(masses) / len(masses)
    mass_std = math.sqrt(sum([(mass - mass_mean)**2
                              for mass in masses]) / len(masses))
    return mass_mean, mass_std
```

Measure new physics

```python
def step(bothons, E_split, E_mass):
    new_bothons = []
    for bothon in bothons:
        if bothon > E_split:
            # Bothon is above splitting energy. Split it into
            # two new bothons and store their energies.
            new_energy = (bothon - 2 * E_mass) / 2
            new_bothons.extend([new_energy, new_energy])
        else:
            # Double bothon energy
            new_bothons.append(bothon * 2)
    return new_bothons

def solution(initial_state, E_split, E_mass, stop_threshold):
    bothons = initial_state
    steps = 0

    while (steps := steps + 1)  <= stop_threshold:
        bothons = step(bothons, E_split, E_mass)

    number_of_bothons = len(bothons)
    return number_of_bothons
```

Markov chain Shakespeare sonnets

```python
def solution(sonnets):
    # Note! This is definitely not the most reasonable algorithm to use!

    # Setup dictionary
    word_dict = dict()

    # Setup variables to store words
    w1, w2, w3 = "", "", ""

    # Setup boolean to track whitespace
    in_space = False

    # Loop over the text character by character
    for char in sonnets:
        if char.isspace():
            # Character is whitespace: we are in a space between words
            in_space = True
            # Keep going until we find the start of a new word
            continue

        else:
            # Character is not whitespace, i.e. part of a word
            if in_space:
                # The previous character was whitespace
                # We have found a new word
                in_space = False

                if w1 != "" and w2 != "" and w3 != "":
                    # All of w1, w2 and w3 are words

                    if (w1, w2) not in word_dict.keys():
                        # The dictionary does not yet have the key (w1, w2)
                        word_dict[(w1, w2)] = list()

                    # Add w3 to the list of words that can follow (w1, w2)
                    word_dict[(w1, w2)].append(w3)

                # Move w2->w1, w3->w2 and reset w3 so it can start
                # collecting characters from the next word when a
                # non-space character is found
                w1, w2, w3 = w2, w3, ""

            # Add this character to w3
            w3 += char

    # Add the last w3 to the dictionary. This is necessary since in the
    # loop, words are only added when the start of the next word is found.
    # Therefore, the last word will not be added while looping.
    if (w1, w2) not in word_dict.keys():
        word_dict[(w1, w2)] = list()
    word_dict[(w1, w2)].append(w3)

    return word_dict
```

Exercise 1

```python
## -------------------EXERCISE DESCRIPTION------------------ ##
## Count the number of times a given number appears in a     ##
## list of integers and return the result.                   ##
##                                                           ##
## Inputs:                                                   ##
## lst: list                                                 ##
## number: int                                               ##
##                                                           ##
## Returns:                                                  ##
## count: int                                                ##
##                                                           ##

def solution(lst, number):
    count = 0
    for element in lst:
        if element == number:
            count += 1
    return count
```

Count instances

Normalize values

Clamp matrix values

n largest numbers

Data processing

Measure new physics

Markov chain Shakespeare sonnets

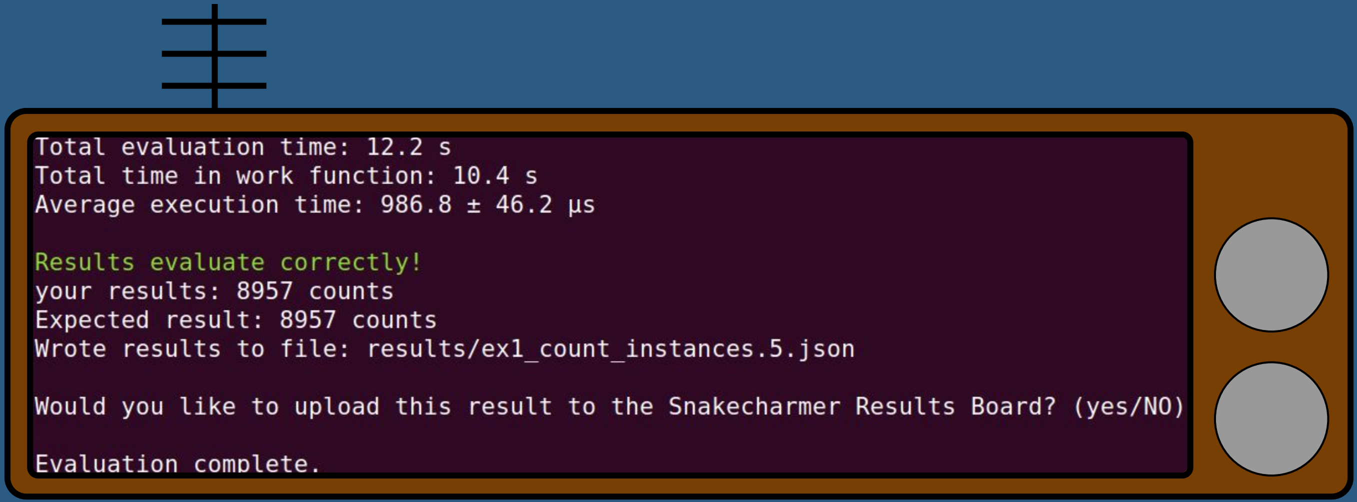
Infrastructure

> python test_run.py ex1_count_instances.py

Infrastructure

> python test_run.py ex1_count_instances.py

> python evaluate.py ex1_count_instances.py



Snakecharming Results

Last updated: 09:20:27

Exercise 1 | Exercise 2 | Exercise 3 | Exercise 4 | Exercise 5 | Exercise 6 | Exercise 7

All | numpy | no-built-ins | python-3.11 | non-SWAN | numba-njit | no-imports | MY_FLAIR | MY_OTHER_FLAIR

User	Runtime	Timestamp	Code	Flair
Beth-99	52.6 ± 64.4 μs	17:36:48	View code	numpy no-built-ins python-3.11 non-SWAN
theo_d	58.4 ± 33.6 μs	17:25:57	View code	numpy python-3.11
snakeslayer	59.2 ± 38.0 μs	17:25:43	View code	numpy no-built-ins python-3.11
Ian-Evan	60.4 ± 38.2 μs	17:40:13	View code	numpy no-built-ins python-3.11
vkucera	65.7 ± 15.4 μs	17:37:41	View code	numpy numba-njit no-built-ins python-3.11
Alex	84.7 ± 79.9 μs	17:33:05	View code	numpy no-built-ins python-3.11
lbozianu	407.1 ± 34.6 μs	17:28:22	View code	numpy no-built-ins python-3.11
cversteg	1415.9 ± 146.8 μs	17:21:14	View code	no-imports no-built-ins python-3.11
SWANdrzej	2762.8 ± 60.7 μs	18:51:12	View code	no-imports python-3.11
	5221.1 ± 56.8 μs	18:09:11	View code	numpy no-built-ins python-3.11
Beth-99	8539.3 ± 2756.6 μs	17:28:06	View code	no-imports no-built-ins python-3.11 non-SWAN
anonymous	15911.2 ± 156.2 μs	18:04:53	View code	MY_FLAIR MY_OTHER_FLAIR no-imports no-built-ins python-3.11
Coldstream	21247.1 ± 596.5 μs	17:32:27	View code	numpy no-built-ins python-3.11

Snakecharming Results

Last updated: 09:20:27

Exercise 1 | Exercise 2 | Exercise 3 | Exercise 4 | Exercise 5 | Exercise 6 | Exercise 7

All | numpy | no-built-ins | python-3.11 | non-SWAN | numba-njit | no-imports | MY_FLAIR | MY_OTHER_FLAIR

User	Runtime	Timestamp	Code	Flair
Beth-99	52.6 ± 64.4 μs	17:36:48	View code	numpy no-built-ins python-3.11 non-SWAN
theo_d	58.4 ± 33.6 μs	17:25:57	View code	numpy python-3.11
snakeslayer	59.2 ± 38.0 μs	17:25:43	View code	numpy no-built-ins python-3.11
Ian-Evan	60.4 ± 38.2 μs	17:40:13	View code	numpy no-built-ins python-3.11
vkucera	65.7 ± 15.4 μs	17:37:41	View code	numpy numba-njit no-built-ins python-3.11
Alex	84.7 ± 79.9 μs	17:33:05	View code	numpy no-built-ins python-3.11
lbozianu	407.1 ± 34.6 μs	17:28:22	View code	numpy no-built-ins python-3.11
cversteg	1415.9 ± 146.8 μs	17:21:14	View code	no-imports no-built-ins python-3.11
SWANdrzej	2762.8 ± 60.7 μs	18:51:12	View code	no-imports python-3.11
	5221.1 ± 56.8 μs	18:09:11	View code	numpy no-built-ins python-3.11
Beth-99	8539.3 ± 2756.6 μs	17:28:06	View code	no-imports no-built-ins python-3.11 non-SWAN
anonymous	15911.2 ± 156.2 μs	18:04:53	View code	MY_FLAIR MY_OTHER_FLAIR no-imports no-built-ins python-3.11
Coldstream	21247.1 ± 596.5 μs	17:32:27	View code	numpy no-built-ins python-3.11

Goals

- ➔ considering options
- ➔ results with different merits
- ➔ success through iteration, reflection

[Link to the exercises]