Homework No. 01

Due: 23:59, 19 February, 2025

Max points: 100

Rules

- **No late homeworks.** A penalty of 10 points is applied for each day.
- **No plagiarism.** Collaboration is encouraged, but copying someone else's work without proper attribution is not admitted and invalidates the submission. A penalty is applied to all parties included.
- **Responsible AI Use.** AI assistants (e.g. ChatGPT) may be used as a learning tool, but the primary goal of homework is to develop your own problem-solving and coding skills. AI should be used minimally and responsibly with proper understanding and attribution. Submissions that rely excessively on AI without demonstrating personal effort may receive penalties.

Submission procedure

- A single Jupyter Notebook file can be used. The following naming convention should be used: `homework{number}_{student name}.ipynb`. For example, `homework01_John_Doe.ipynb`.
- At the start of the file, homework number and student full name should be mentioned. Problem solutions should be clearly separated by problem numbers. For example:

```
"""
# Homework 01
## Name: Jane Dane

### Problem 1

...

### Problem 2

...
"""
```

- Solution files should be uploaded to YSU Moodle. Alternatively, you can commit your solutions to a Git repository and provide the repository URL on Moodle.

Problem 1 [10 points]

Write a function `make_polynomial(*coefficients)` that takes an arbitrary number of coefficients and returns a function representing the polynomial. The returned function should compute the polynomial's value when called with a specific x.

Example

```
def make_polynomial(*coefficients):
    pass

poly = make_polynomial(2, 3, 5)  # Represents 2 + 3x + 5x^2
print(poly(0))  # 2
print(poly(1))  # 10
```

Problem 2 [10 points]

Write a function that calculates the n-th derivative of a polynomial. The polynomial can be represented as a list of coefficients, where the index corresponds to the power of x. For example, $[3, 1, 2]$ represents the polynomial $3 + x + 2x^2$.

Example

```
def polynomial_nth_derivative(coefficients, n):
    pass

print(nth_derivative([3, 1, 2], 1))  # [1, 4] (Derivative of 3 + x + 2x^2 is 4x)
print(nth_derivative([3, 1, 2], 2))  # [4] (Second derivative is 4)
print(nth_derivative([3, 1, 2], 3))  # [0] (Third derivative is 0)
```

Problem 3 [10 points]

Write a function `matrix_power(matrix, n)` that computes the n-th power of a given square matrix.

- Assume n is a non-negative integer.
- If $n = 0$, return the identity matrix of the same size.
- If $n = 1$, return the matrix itself.
- For $n > 1$, compute the matrix product repeatedly.

Example

```python
def matrix_power(matrix, n):
    pass

matrix = [
    [1, 2],
    [3, 4]
]

print(matrix_power(matrix, 3)) # [[37, 54], [81, 118]]
print(matrix_power(matrix, 0)) # [[1, 0], [0, 1]]
```

Problem 4 [10 points]

Write a function `compose(*funcs)` that takes an arbitrary number of single-argument functions and returns a new function that is the composition of the input functions. The composed function should apply each function in the order they were passed.

Example

```python
def double(x):
    return x * 2

def increment(x):
    return x + 1

def square(x):
    return x * x

def compose(*funcs):
    pass

composed = compose(square, increment, double)
print(composed(3))  # square(increment(double(3))) = 49
```

Problem 5 [10 points]

Write a Python recursive function to generate all possible combinations of a set of elements.

Note: This will be your implementation of `itertools.combinations` function.
Note: It is not required, but this function can be a generator function.

Example

```
def generate_combinations(elements, k):
    pass

elements = [1, 2, 3, 4]
k = 3
combinations = generate_combinations(elements, k)
print(combinations) # [(1, 2, 3), (1, 2, 4), (1, 3, 4), (2, 3, 4)]
```

Problem 6 [10 points]

A perfect number is a positive integer that is equal to the sum of its positive divisors, excluding the number itself. For example, 6 is a perfect number.

Write a Python generator function that generates all the perfect numbers up to a given limit.

Example

```
def generate_perfect_numbers(limit):
    pass

for num in perfect_numbers(100):
    print(num, end=" ") # 6 28
```

Problem 7 [10 points]

An Armstrong number is a number that is the sum of its own digits each raised to the power of the number of digits. For example, 153 is an Armstrong number as $153 = 1^3 + 5^3 + 3^3$.

Write a Python generator function that generates all the Armstrong numbers up to a given limit.

Example

```
def generate_armstrong_numbers(limit):
    pass

for num in generate_armstrong_numbers(1000):
    print(num, end=" ") # 1 2 3 4 5 6 7 8 9 153 370 371 407
```

Problem 8 [10 points]

Note: The following problem can be solved using generator functions in the Python standard library.

Write a Python function that takes a list of numbers and returns a list of all the triples of numbers in the list that form a Pythagorean triplet.

Example

```python
def pythagorean_triplets(numbers):
    pass

print(pythagorean_triplets([3, 4, 5, 6, 7, 8, 9, 10])) # [(3, 4, 5), (6, 8, 10)]
```

Problem 9 [10 points]

Write a Python decorator function that caches the output of a function. It should return the cached value if the function is called again with the same arguments. Provide an example usage of the decorator.

Example

```
def cache(func):
    pass

@cache
def fibonacci(n):
    pass

print(fibonacci(10)) # 55
print(fibonacci(10)) # 55 (this is a cached value)
```

Problem 10 [10 points]

Write a Python decorator function that limits the number of times a function can be called. Provide an example usage of the decorator.

Example

```
def limit_calls(max_calls):
    pass

@limit_calls(3)
def greet():
    print("Hello world!")

greet() # Hello world!
greet() # Hello world!
greet() # Hello world!
greet() # Function `greet` can only be called 3 times.
```