

# Debugging and File Processing

## Bug Swatting

Your tutors have tried to write a program that finds the first `n` prime numbers, for an integer `n`. Unfortunately, we're all noobs. Our code has several bugs in it and we need some help. The code is shown below; your task is to find and fix the errors. This code can also be downloaded as [week05\\_primes.py](#).

```
def is_prime(n) :
    """Returns True iff 'n' is prime.

    Parameters:
        n (int): Integer value to be tested to see if it is prime.

    Return:
        bool: True if 'n' is prime. False otherwise.

    Preconditions:
        n > 1
    """
    for i in range(2, n) :
        # Check if i is a factor of n
        if n % i == 0 :
            return False
        else :
            return True

def get_primes(n) :
    """Return a list of the first n primes.

    Parameters:
        n (int): Number of prime numbers to find.

    Return:
        list[int]: The first 'n' prime numbers.

    Preconditions:
        n > 0
    """
    primes = []
    i = 2
    while len(primes) < n :
        if is_prime(i) :
            primes.append(i)
            i += 1

n = int(input("How many primes? "))
print("The first" + n + "primes are:", primes)
```

The following example shows how the program should work:

```
How many primes? 7
The first 7 primes are: [2, 3, 5, 7, 11, 13, 17]
```

# Function Finder

In this task, we will write a function that reads and analyses Python source code files. In particular, the function we write will find all function definitions within the file.

1. Write a function `find_functions(filename)` that takes the name of a file containing Python code, and finds the name of each function defined in the file. The output should be written to a file called `functions.txt`, with one function name per line. You may like to use the sample file `week05_functions.py` to test your code. For this file, the output file `functions.txt` should contain:

```
def square(x):
def add(x, y):
def get_0():
```

2. Modify this function to instead return a list of tuples of the form `(linenum, name, args)`, where `name` is the name of a function, `args` is a tuple of parameter names to the function, and `linenum` is the line number containing the function definition. For example:

```
>>> find_functions('week05_functions.py')
[(1, 'square', ('x',)),
 (5, 'add', ('x', 'y')),
 (14, 'get_0', ())]
```

## Challenge: Extract and Parse Function Comment

Modify this function to parse the function's comment and include it in the tuple that represents a function. Make use of the Python convention for function comments. Tackle this problem in four stages.

1. Extract only the first line of text after the opening `"""` as the summary comment. Include this as a string in the tuple representing a function, `(linenum, name, args, summary_comment)`.
2. If the closing `"""` for the comment is not on the same line as the opening `"""`, assume that there is a blank line and then read the following lines until the closing `"""`. This makes up the details of the comment. Create a tuple that includes the summary and the details as two strings, `(summary, details)`; and include this in the tuple representing a function, `(linenum, name, args, comment)`.
3. Now, for a further challenge, based on the Google Python commenting style, as demonstrated in lectures; extract the description of the parameters, return value and preconditions from the function comment. Put these as separate entries in the tuple that represents the comment, `(summary, [parameter_descriptions], return, preconditions)`. (Each parameter description is on a separate line, so may be a list of strings.) Include this comment tuple in the tuple representing a function, `(linenum, name, args, comment)`.
4. Now create a tuple that is the parameter name and the comment describing the parameter, `(parameter_name, comment)`. In the tuple that represents a function, replace the tuple of parameter names with a list of tuples that has the parameter name and descriptive comment, `(linenum, name, [args], comment)`. The rest of the function comment can still be included in the comment tuple as: `(summary, return, preconditions)`.

Test your code using some of the code you have written and the lecture examples.

```

>>> find_functions('week04b-grade_book.py')
[(156, 'calculate_percentage', [('results', 'results (list): Results for all
assessment items.']), ('Calculate the percentage achieved based on these
results.', 'float: Percentage based on assessment item results & their
weights.')),

(173, 'process_results', [('results', 'results (list): List of each
student's results in each course.']), ('Calculate grades for students based
on their results in courses.', 'list: List of final grades of each student
in each course:')),

(197, 'get_exam_cap', [('course_code', 'course_code (str): Course code used
to look up exam cap.']), ('Find the exam cap for this course.', 'The caps
required to achieve each grade level in this')),

(215, 'get_grade_cutoffs', [('course_code', 'course_code (str): Course code
used to look up grade cut offs.']), ('Find the grade cut offs for this
course.', 'list: The cut offs for each grade level in this course (7 ...
2).')),

(232, 'final_grade', [('final_mark', 'final_mark (float): Final mark
achieved in a course.'], ('exam_result', 'exam_result (int): Mark achieved
in the final exam for the course.'], ('grade_cutoffs', 'grade_cutoffs
(list): Final mark required for each grade level.'], ('exam_cap', 'exam_cap
(list): Minimum mark required in the exam to achieve')), ('Calculate a
student's final grade for a course.', 'int: Grade level achieved in this
course (7 ... 1).')),

(282, 'output_grades', [('grades', )], ('Simple formatted output of final
results for all students.', )),

(289, 'demo', [], ('Demonstration of functionality.', ))]

```

Note that in the output above, space has been inserted between each of the tuples representing a function. This has been done to make the output more readable. The output from executing the function in Python would not have these spaces.