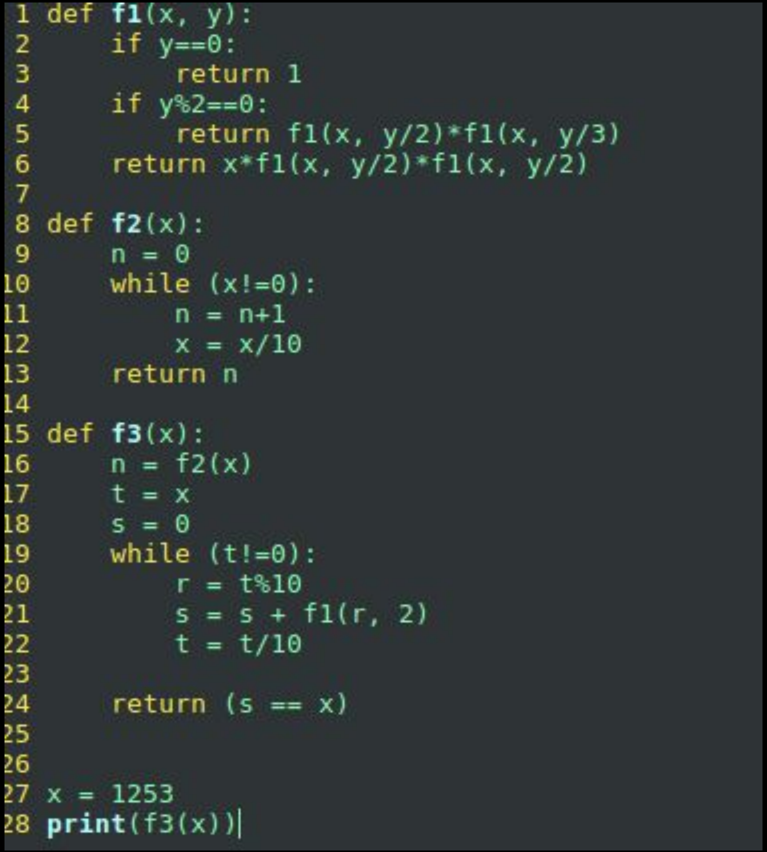


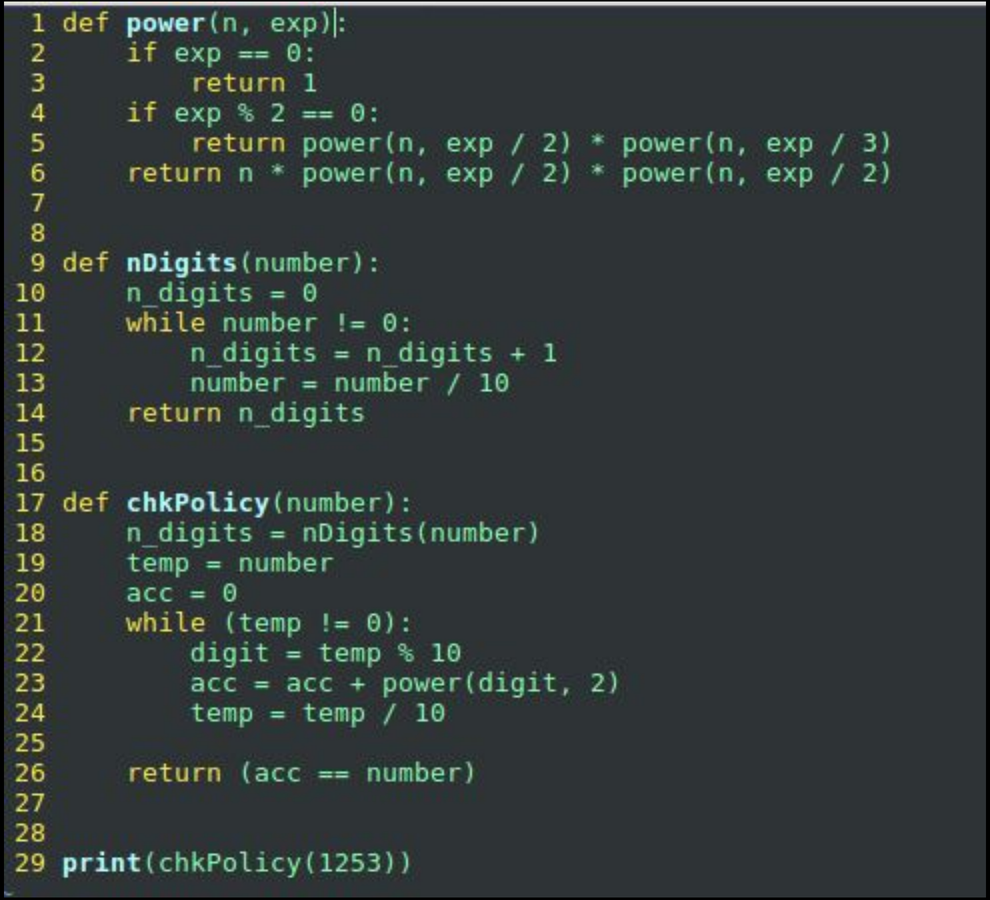
```
def f1(x, y):
    if y==0:
        return 1
    if y%2==0:
        return f1(x, y/2)*f1(x, y/3)
    return x*f1(x, y/2)*f1(x, y/2)

def f2(x):
    n = 0
    while (x!=0):
        n = n+1
        x = x/10
    return n

def f3(x):
    n = f2(x)
    t = x
    s = 0
    while (t!=0):
        r = t%10
        s = s + f1(r, 2)
        t = t/10

    return (s == x)


x = 1253
print(f3(x))
```

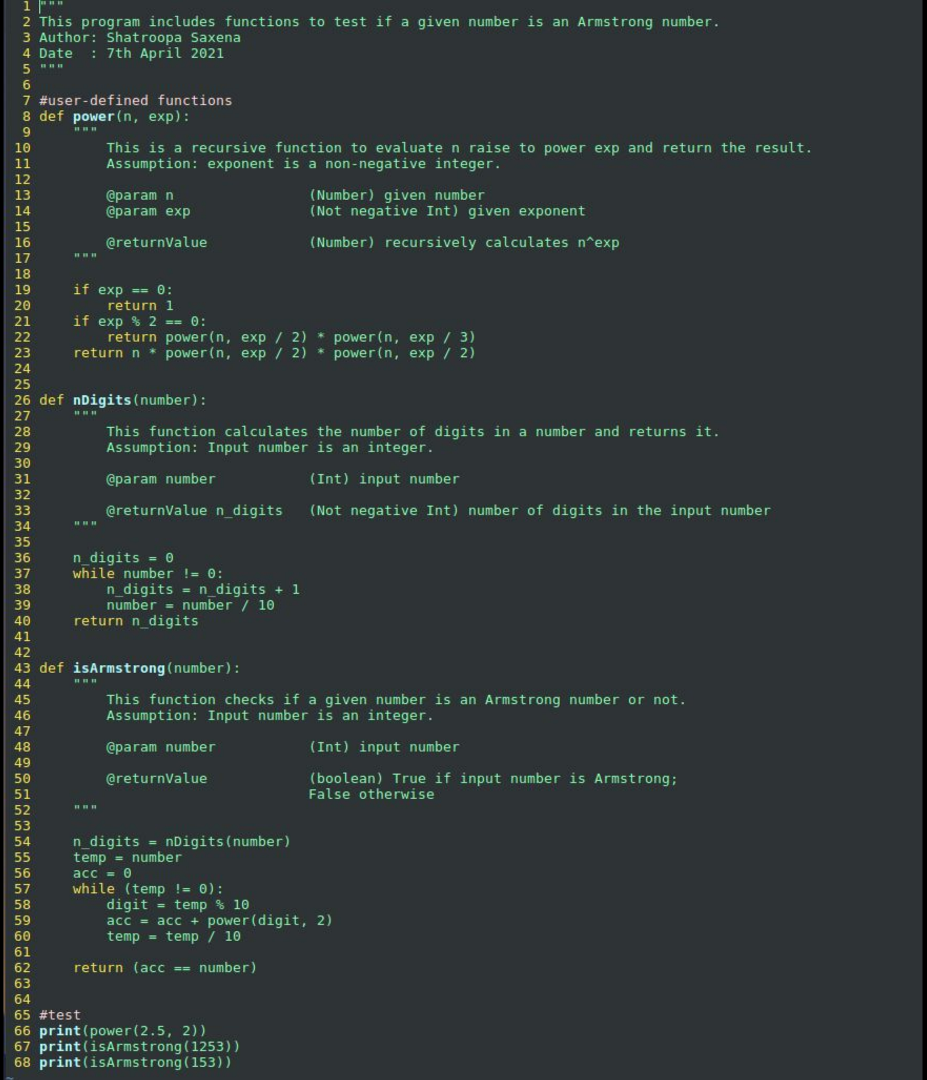

```python
def power(n, exp):
    if exp == 0:
        return 1
    if exp % 2 == 0:
        return power(n, exp / 2) * power(n, exp / 3)
    return n * power(n, exp / 2) * power(n, exp / 2)


def nDigits(number):
    n_digits = 0
    while number != 0:
        n_digits = n_digits + 1
        number = number / 10
    return n_digits


def chkPolicy(number):
    n_digits = nDigits(number)
    temp = number
    acc = 0
    while (temp != 0):
        digit = temp % 10
        acc = acc + power(digit, 2)
        temp = temp / 10

    return (acc == number)


print(chkPolicy(1253))
```



```python
"""
This program includes functions to test if a given number is an Armstrong number.
Author: Shatroopa Saxena
Date  : 7th April 2021
"""

#user-defined functions
def power(n, exp):
    """
        This is a recursive function to evaluate n raise to power exp and return the result.
        Assumption: exponent is a non-negative integer.

        @param n                (Number) given number
        @param exp              (Not negative Int) given exponent

        @returnValue            (Number) recursively calculates n^exp
    """

    if exp == 0:
        return 1
    if exp % 2 == 0:
        return power(n, exp / 2) * power(n, exp / 3)
    return n * power(n, exp / 2) * power(n, exp / 2)


def nDigits(number):
    """
        This function calculates the number of digits in a number and returns it.
        Assumption: Input number is an integer.

        @param number           (Int) input number

        @returnValue n_digits   (Not negative Int) number of digits in the input number
    """

    n_digits = 0
    while number != 0:
        n_digits = n_digits + 1
        number = number / 10
    return n_digits


def isArmstrong(number):
    """
        This function checks if a given number is an Armstrong number or not.
        Assumption: Input number is an integer.

        @param number           (Int) input number

        @returnValue            (boolean) True if input number is Armstrong;
                                False otherwise
    """

    n_digits = nDigits(number)
    temp = number
    acc = 0
    while (temp != 0):
        digit = temp % 10
        acc = acc + power(digit, 2)
        temp = temp / 10

    return (acc == number)


#test
print(power(2.5, 2))
print(isArmstrong(1253))
print(isArmstrong(153))
```

SOFTWARE DOCUMENTATION

Contents

Purpose & Benefits

- Improved understanding of code
- Readability
- Makes your code durable
- Makes it easy to spot bugs
- **Makes your code beautiful !!!**

What constitutes a well-documented software?

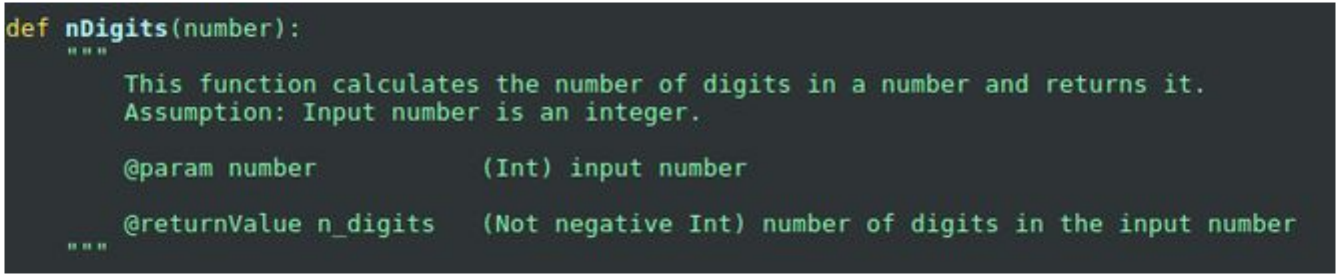
- Standard directory structure
- Consistent naming convention
- Identifier names do matter!
- Optimally commented
- README file
  - Purpose of software
  - Software dependencies (if any)
  - How to execute/ Installation instructions
  - Details of platform on which the software is built
  - Design and Architecture
  - Code Description
  - Tests’ Description (if any)
- Other optional files: Install, License, TODO, Changelog

Documentation in Python

Comments

- Comments clarify the code and they are added with purpose of making the code easier to understand.
- In Python, comments begin with ‘**#**’. Note that these are single - line comments.
- Comments can be block or inline comments.
- Block comments are used to explain a block of code.
- Inline comment is a comment on the same line as a code statement.

docstrings



```
def nDigits(number):
    """
        This function calculates the number of digits in a number and returns it.
        Assumption: Input number is an integer.

        @param number           (Int) input number

        @returnValue n_digits   (Not negative Int) number of digits in the input number
    """
```

- A string literal that occurs as the first statement in a module, function, class, or method definition. Such a docstring becomes the __doc__ special attribute of that object.
- Explains how a particular function can be used and the general purpose of a function, class, or module.
- In Python, docstrings are enclosed between triple quotes: “””….”””
- Can either be one-line or multiline as per the need.
- Can be used to embed more-than-documentation behavior. For eg: such as unit test logic:

```
“””
>>> nDigits(45)
2
“””
```

Standards

- Comments should be complete sentences. The first word should be capitalized, unless it is an identifier that begins with a lower case letter.
- Block comments are indented to the same level as that block. Each line of a block comment starts with a # and a single space.
- In case of paragraphs inside a block comment, separate them by a line containing a single #.
- Use inline comments sparingly. When used they should start with a # and a single space. When necessary, instead of explaining what the code does, explain why you have that line of code in an inline comment.
- Unless the entire docstring fits on a line, place the closing quotes on a new line.
- Triple quotes are used even though the docstring fits on one line instead of ‘#’.

reStructuredText

- easy-to-read, what-you-see-is-what-you-get plaintext markup syntax and parser system.
- Useful for in-line program documentation (such as Python docstrings), for quickly creating simple web pages, and for standalone documents.
- It defines and implements a markup syntax for use in Python docstrings and other documentation domains.

Sphinx

```
pip install -U Sphinx
```

- Tool to create “intelligent and beautiful documentation” .
- Formats: HTML/ PDF (via LaTeX)/ txt (man pages)
- default plain-text markup format: *reStructuredText*
- Well-commented code in rst format is a plus!

References

- Python Documentation: https://docs.python-guide.org/writing/documentation
- Comments: https://www.python.org/dev/peps/pep-0008/#comments
- Docstrings: https://www.python.org/dev/peps/pep-0257/#specification
- reStructuredText: https://docutils.sourceforge.io/rst.html
- Sphinx: https://www.sphinx-doc.org/en/master/

Extra Reading

- Doctest: https://docs.python.org/3/library/doctest.html
- Docutils: https://docutils.sourceforge.io/index.html
- reStructuredText: https://www.sphinx-doc.org/en/master/usage/restructuredtext/index.html
- reStructuredText: https://docutils.sourceforge.io/docs/user/rst/quickref.html
- toctree: https://www.sphinx-doc.org/en/master/usage/restructuredtext/directives.html#toctree-directive
- autodoc: https://www.sphinx-doc.org/en/master/usage/extensions/autodoc.html#module-sphinx.ext.autodoc
- Sphinx tutorial: https://matplotlib.org/sampledoc/