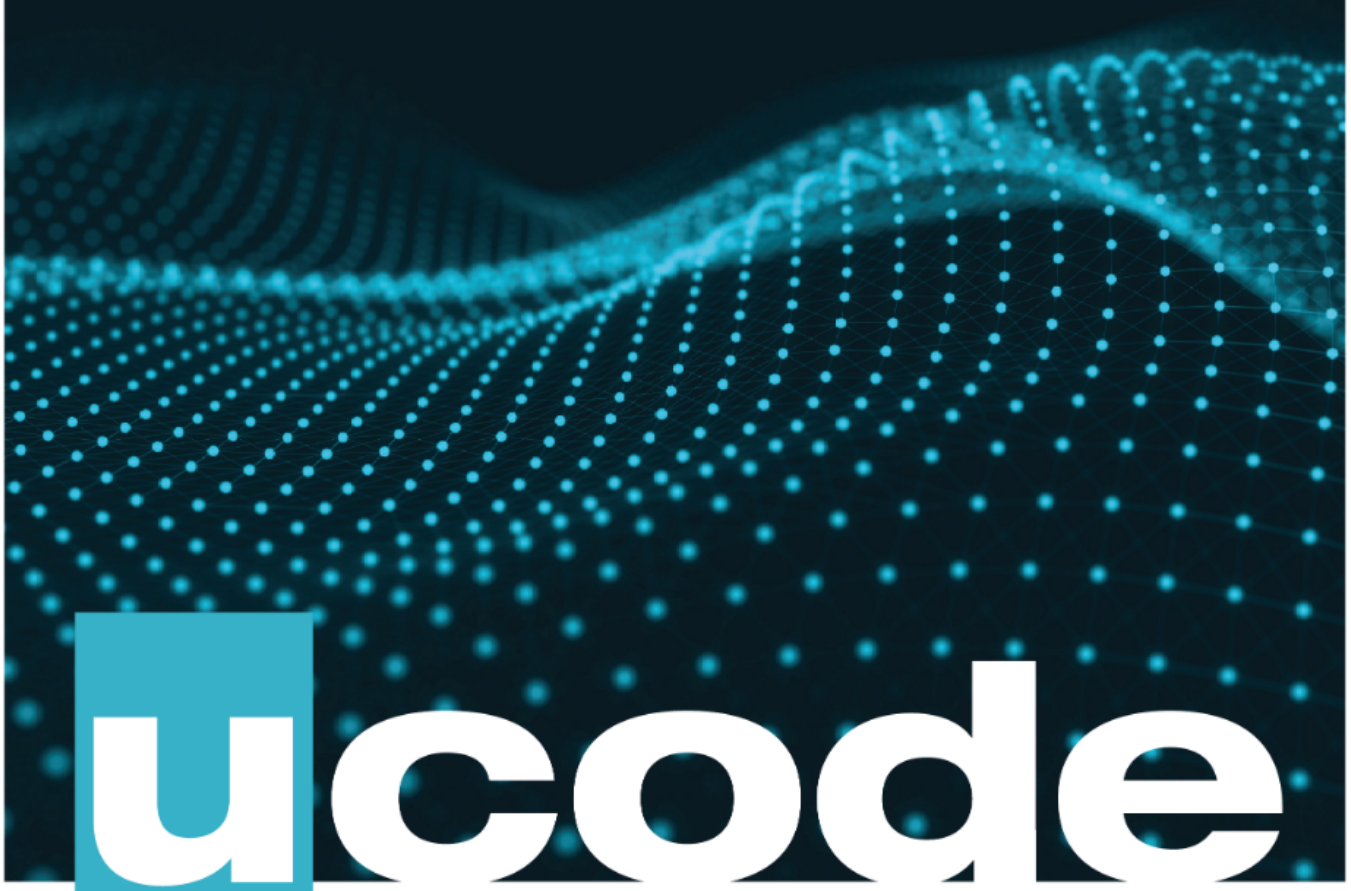
Sprint 01

Marathon Python

April 13, 2021



ucode

Contents

ucode

Engage

DESCRIPTION

```
print('Hello, ucoder!')
```

`Functions` are a convenient way to divide code into useful and ordered blocks, make it more readable, reuse it and save some time.

Python contains many built-in functions like `print()`, `len()`, etc., but you can also create your own functions. These functions are called user-defined functions.
In this challenge, you have to create a variety of functions.

BIG IDEA

Functions.

ESSENTIAL QUESTION

What are functions used for?

CHALLENGE

Learn to create functions in Python.

Investigate

GUIDING QUESTIONS

We invite you to find answers to the following questions. By researching and answering them, you will gain the knowledge necessary to complete the challenge. To find answers, ask the students around you and search the internet. We encourage you to ask as many questions as possible. Note down your findings and discuss them with your peers.

- What is a function in programming?
- What are functions used for? What is their role?
- What are the types of functions in Python?
- What is a loop in programming?
- What loops are there in Python?
- What are loops needed for?

GUIDING ACTIVITIES

Complete the following activities. Don't forget that you have a limited time to overcome the challenge. Use it wisely. Distribute tasks correctly.

- Read about functional programming in Python.
- Find information about imperative programming.
- Read information about functions: how to work with them, what functions are, etc. For example, read this article on functions in Python.
- Learn information about loops.
- Attentively watch and investigate learning videos available on the challenge page. Try to repeat all actions.
- Clone your git repository issued on the challenge page in the LMS.
- Proceed with tasks.

ANALYSIS

Analyze your findings. What conclusions have you made after completing guiding questions and activities? In addition to your thoughts and conclusions, here are some more analysis results.

- Be attentive to all statements of the story.
- All tasks are divided into Act Basic and Act Advanced. You need to complete all basic tasks to validate the Sprint. But to achieve maximum points, consider accomplishing advanced tasks also.
- Analyze all information you have collected during the preparation stages. Try to define the order of your actions.
- Perform only those tasks that are given in this document.
- Submit only those files that are described in the story. Only useful files allowed, garbage shall not pass!
- Run the scripts using `python3`.

- Make sure that you have `Python` with a `3.8` version, or higher.
- Use the standard library available after installing `Python`. You may use additional packages/libraries that were not previously installed only if they are specified in the task.
- To figure out what went wrong in your code, use PEP 553 -- Built-in breakpoint().
- Complete tasks according to the rules specified in the PEP8 conventions.
- The solution will be checked and graded by students like you. Peer-to-Peer learning.
- Also, the challenge will pass automatic evaluation which is called `Oracle`.
- If you have any questions or don't understand something, ask other students or just Google it.

Act Basic: Task 00

NAME

What is your name?

DIRECTORY

`t00_what_is_your_name/`

SUBMIT

`print_filename.py`

LEGEND

BRIDGEKEEPER: Stop. What... is your name?
SIR GALAHAD: Sir Galahad of Camelot.
BRIDGEKEEPER: What... is your quest?
SIR GALAHAD: I seek the Grail.
BRIDGEKEEPER: What... is your favourite colour?
SIR GALAHAD: Blue-no! *[he is also thrown over the edge]* YEELLLLLLLLLLLLOOOOOOOOWWWWWWW!
-- Monty Python and the Holy Grail

BEFORE YOU BEGIN

In order to complete this task, you must be able to answer the following questions:

- What is a function in Python?
- How to define a function?
- Which keyword denotes the beginning of a function header?
- What is the purpose of this statement: `if __name__ == '__main__':` ?
- What is the role of using `__file__` ?
- How to get the basename from a pathname?

DESCRIPTION

Create a script that contains and calls a function `print_filename()` .
The function must print the base name of the running script file using the `__file__` variable.

CONSOLE VIEW

```
>python3 print_filename.py
print_filename.py
>
```

PYTHON INTERPRETER

```
>python3
>>> from print_filename import print_filename
>>> print_filename()
print_filename.py
>>>
```

SEE ALSO

Python functions
What is if __name__ == "__main__"?
__file__ (A Special variable) in Python
os.path.basename() method

Act Basic: Task 01

NAME

Arguments

DIRECTORY

`t01_arguments/`

SUBMIT

`crystal_ball.py`

BEFORE YOU BEGIN

In order to complete this task, you must be able to answer the following questions:

- What is a function argument?
- How to pass multiple arguments to a function?
- What will happen if a function expects two arguments, and you only pass one?
- What are the logical operators `and` and `or` used for?

Below is a simple example of passing an argument to a function.

```
>python3
>>> def my_function(number_of_legs):
...     print(f'I have {number_of_legs} legs!')
...
>>> my_function(8)
I have 8 legs!
>>>
```

DESCRIPTION

Create a function `crystal_ball()` that takes two arguments: `courage` and `intelligence`. The function must print a message depending on the values of these two numbers. The conditions are:

- if both variables are greater than 50:
  `'I see great success in your future.'`
- otherwise, if `courage` is greater or equal to 100, or `intelligence` is lower or equal to 10, then:
  `'Your life is in danger!'`
- in all other cases:
  `'Your future is a mystery...'`

The script in the EXAMPLE tests your function. If everything is correct, it should generate output as seen in the CONSOLE VIEW. Pay attention that you must only submit the file crystal_ball.py, not the test script.

EXAMPLE

```python
# s01t01_arguments_main.py
from crystal_ball import crystal_ball


def print_result(courage, intelligence):
    print(f'Reading the future for an adventurer with {courage} courage '
          f'and {intelligence} intelligence...')
    crystal_ball(courage, intelligence)
    print('***')

if __name__ == '__main__':
    print_result(19, 0)
    print_result(57, 60)
    print_result(200, 79)
    print_result(150, 15)
    print_result(30, 100)
    print_result(100, 25)
    print_result(50, 55)
    print_result(50, 9)
```

CONSOLE VIEW

```
>python3 s01t01_arguments_main.py
Reading the future for an adventurer with 19 courage and 0 intelligence...
Your life is in danger!
***
Reading the future for an adventurer with 57 courage and 60 intelligence...
I see great success in your future.
***
Reading the future for an adventurer with 200 courage and 79 intelligence...
I see great success in your future.
***
Reading the future for an adventurer with 150 courage and 15 intelligence...
Your life is in danger!
***
Reading the future for an adventurer with 30 courage and 100 intelligence...
Your future is a mystery...
***
Reading the future for an adventurer with 100 courage and 25 intelligence...
Your life is in danger!
***
Reading the future for an adventurer with 50 courage and 55 intelligence...
Your future is a mystery...
***
Reading the future for an adventurer with 50 courage and 9 intelligence...
Your life is in danger!
```

```
***
>
```

PYTHON INTERPRETER

```
>python3
>>> from crystal_ball import crystal_ball
>>> crystal_ball(15, 3)
Your life is in danger!
>>> crystal_ball(64, 180)
I see great success in your future.
>>>
```

SEE ALSO

Understanding Boolean Logic in Python 3

Act Basic: Task 02

NAME

Return

DIRECTORY

`t02_return/`

SUBMIT

`shop.py`

BEFORE YOU BEGIN

In order to complete this task, you must be able to answer the following questions:

- How to return a value from a function?
- What happens when you don't return anything?
- How to use default arguments in a function?
- What is the advantage of using default arguments?

DESCRIPTION

Create a script with two functions: `buy_milk()` and `buy_bread()` that will exchange money for product. The purpose of both of these functions is to return to you a certain amount of product based on how much money you have.

Here are the requirements:

- both functions must:
  - take an argument `money`
  - if `money` was not provided, set it to `0`
  - contain a variable `product` with the value: `'[milk]'` for `buy_milk()` or `'[bread]'` for `buy_bread()`
  - contain a variable `price` with the value: `25` for `buy_milk()` or `19` for `buy_bread()`
  - if `money` is greater or equal to the `price`, then return the `product` multiplied by how many items can be bought with this amount of money (e.g., if the product costs 25, and there's 100 money, multiply the product by 4)
  - if there isn't enough money, don't return anything
- in addition, the function `buy_bread()` has a limit of three breads to be sold at once, so the maximum multiplier is three

The script in the EXAMPLE tests your function. If everything is correct, it should generate output as seen in the CONSOLE VIEW. Pay attention that you must only submit the file shop.py, not the test script.

EXAMPLE

```python
# s01t02_return_main.py
from shop import buy_milk, buy_bread

if __name__ == '__main__':
    # money = price of 1 item
    print(f'Buy milk with $25, result: {buy_milk(25)}')
    print(f'Buy bread with $19, result: {buy_bread(19)}')
    # no money
    print(f'Buy milk with nothing, result: {buy_milk()}')
    print(f'Buy bread with nothing, result: {buy_bread()}')
    # a lot of money
    print(f'Buy milk with $200, result: {buy_milk(200)}')
    print(f'Buy bread with $200, result: {buy_bread(200)}')
    # money < price of item
    print(f'Buy milk with $10, result: {buy_milk(10)}')
    print(f'Buy bread with $10, result: {buy_bread(10)}')
    # more tests
    print(f'Buy milk with $53, result: {buy_milk(53)}')
    print(f'Buy bread with $53, result: {buy_bread(53)}')
    print(f'Buy milk with $100, result: {buy_milk(100)}')
    print(f'Buy bread with $100, result: {buy_bread(100)}')
```

CONSOLE VIEW

```
>python3 s01t02_return_main.py
Buy milk with $25, result: [milk]
Buy bread with $19, result: [bread]
Buy milk with nothing, result: None
Buy bread with nothing, result: None
Buy milk with $200, result: [milk][milk][milk][milk][milk][milk][milk][milk]
Buy bread with $200, result: [bread][bread][bread]
Buy milk with $10, result: None
Buy bread with $10, result: None
Buy milk with $53, result: [milk][milk]
Buy bread with $53, result: [bread][bread]
Buy milk with $100, result: [milk][milk][milk][milk]
Buy bread with $100, result: [bread][bread][bread]
>
```

PYTHON INTERPRETER

```
>python3
>>> from shop import buy_milk, buy_bread
>>> buy_milk(15)
>>> buy_milk(175)
'[milk][milk][milk][milk][milk][milk][milk]'
>>> buy_bread(1000)
'[bread][bread][bread]'
>>>
```

SEE ALSO

Python return statement
Default arguments in Python

Act Basic: Task 03

NAME

Give it a try

DIRECTORY

`t03_give_it_a_try/`

SUBMIT

`patoi.py`

BEFORE YOU BEGIN

In order to complete this task, you must be able to answer the following questions:

- What types of errors can happen in programming?
- How important is it to catch programming errors?
- What role do `try ... except` blocks play? How to use them?
- What is a `ValueError` in Python?
- How does Python handle a `ValueError`?

DESCRIPTION

Create a script that contains a function called `patoi()`.
The function must take an argument, cast it to `int`, and return the result.
If the casting attempt fails, the function must return `False`.

The script in the EXAMPLE tests your function. Use this as an example, add your own values, and also test your function as shown in the PYTHON INTERPRETER. If everything is correct, it should generate output as shown below. Pay attention that you must only submit the file patoi.py, not the test script, and Oracle will check your function with random values.

EXAMPLE

```
# s01t03_give_it_a_try_main.py
from patoi import patoi


if __name__ == '__main__':
    print(patoi(3))
    print(patoi('Romanes eunt domus'))
    print(patoi('34b'))
    print(patoi(-234.59))
    print(patoi('-234'))
```

CONSOLE VIEW

```
>python3 s01t03_give_it_a_try_main.py
3
False
False
-234
-234
>
```

PYTHON INTERPRETER

```
>python3
>>> from patoi import patoi
>>> patoi(3)
3
>>> patoi('Romanes eunt domus')
False
>>> patoi('34b')
False
>>> patoi(-234.59)
-234
>>> patoi('-234')
-234
>>>
```

SEE ALSO

Errors and Exceptions

Act Basic: Task 04

NAME

Make a list

DIRECTORY

`t04_make_a_list/`

SUBMIT

`list_maker.py`

BEFORE YOU BEGIN

In order to complete this task, you must be able to answer the following questions:

- What is a `list` in Python?
- What are the similarities and differences of strings and lists?
- Do all elements of a list have to be of the same data type?
- How to create a list from a string with these two different results (both ways are just one line of code):
  - `'Hi there'` becomes `['H', 'i', ' ', 't', 'h', 'e', 'r', 'e']`
  - `'Hi there'` becomes `['Hi', 'there']` (Hint: it's a method of `str`)
- How to append a value to a list?
- How to add two lists?
- How to make a string out of a list?

Get started with lists by completing the following simple instructions:

- Create an empty list.
- Create a list with three elements of different data types.
- Print the element at index `1` of your list.
- Print the number of elements in your list.

DESCRIPTION

Create a function `list_maker()` that makes a list out of a string by dividing the string by a given delimiter. The function takes two strings as arguments: a line that will be turned into a list, and a delimiter string, and returns a list. If the given delimiter is an empty string, use a space `' '` as a delimiter instead.

The script in the EXAMPLE tests your function. Use this as an example, add your own values, and also test your function as shown in the PYTHON INTERPRETER. If everything is correct, it should generate output as shown below. Pay attention that you must only submit the file `list_maker.py`, not the test script, and Oracle will check your function with random values.

Note: there is a string method that you can use to divide a string into a list by a certain delimiter.

EXAMPLE

```
# s01t04_make_a_list_main.py
from list_maker import list_maker


if __name__ == '__main__':
    line, delim = 'surprise', '-'
    res = list_maker(line, delim)
    print(f'Our chief weapon is {res}'
          f'... surprise and fear... fear and surprise...')
    # joining the result back into a string using the delimiter
    # must create a string identical to the original one
    if delim.join(res) != line:
        print('Error.')

    line, delim = 'fear;surprise', ';'
    res = list_maker(line, delim)
    print(f'Our {len(res)} weapons are {res} and ruthless efficiency.')
    if delim.join(res) != line:
        print('Error.')

    line, delim = 'fearblasurpriseblaruthless efficiency', 'bla'
    res = list_maker(line, delim)
    print(f'Our {len(res)} weapons are {res},\n\tand '
          f'an almost fanatical devotion to the Pope.')
    if delim.join(res) != line:
        print('Error.')

    line += 'blaan almost fanatical devotion to the Pope'
    res = list_maker(line, delim)
    print(f'Our {len(res)}, no... Amongst our weaponry are such elements as:\n'
          f'\t{res},\n\tand nice red uniforms - Oh damn!\n')
    if delim.join(res) != line:
        print('Error.')

    line, delim = 'abracadabra', 'b'
    res = list_maker(line, delim)
    print(f'Original string: `{line}`; delimiter: `{delim}`;\n\tresult: {res}')
    if delim.join(res) != line:
        print('Error.')

    line, delim = '', '*'
    res = list_maker(line, delim)
    print(f'Original string: `{line}`; delimiter: `{delim}`;\n\tresult: {res}')
    if delim.join(res) != line:
        print('Error.')

    line, delim = 'testing.extra.delims.in.the.end..', '.'
    res = list_maker(line, delim)
    print(f'Original string: `{line}`; delimiter: `{delim}`;\n\tresult: {res}')
```

```
    if delim.join(res) != line:
        print('Error.')
```

CONSOLE VIEW

```
>python3 s01t04_make_a_list_main.py
Our chief weapon is ['surprise']... surprise and fear... fear and surprise...
Our 2 weapons are ['fear', 'surprise'] and ruthless efficiency.
Our 3 weapons are ['fear', 'surprise', 'ruthless efficiency'],
        and an almost fanatical devotion to the Pope.
Our 4, no... Amongst our weaponry are such elements as:
        ['fear', 'surprise', 'ruthless efficiency', 'an almost fanatical devotion to the
↪  Pope'],
        and nice red uniforms - Oh damn!

Original string: `abracadabra`; delimiter: `b`;
        result: ['a', 'racada', 'ra']
Original string: ``; delimiter: `*`;
        result: ['']
Original string: `testing.extra.delims.in.the.end..`; delimiter: `.`;
        result: ['testing', 'extra', 'delims', 'in', 'the', 'end', '', '']
>
```

PYTHON INTERPRETER

```
>python3
>>> from list_maker import list_maker
>>> result = list_maker('idogieatidogiworld', 'i')
>>> result
['', 'dog', 'eat', 'dog', 'world']
>>> type(result)
<class 'list'>
>>> list_maker('', ',')
['']
>>> list_maker('hello', '')
['hello']
>>> list_maker('hello world', '')
['hello', 'world']
>>>
```

SEE ALSO

Python List - A Beginners Guide

Act Basic: Task 05

NAME

While

DIRECTORY

`t05_while/`

SUBMIT

`bookshelf.py`

BEFORE YOU BEGIN

One of the key concepts in programming is iteration - repetition of steps until a condition is met.
In order to complete this task, you must be able to answer the following questions:

- What is a loop in programming?
- What is an exit condition?
- What is an infinite loop?
- How to update a counter variable in a `while` loop?

In this task you will be working with a `while` loop. It's well suited for situations when you don't know how many iterations of the loop you will need.

The basic syntax of a `while` loop in Python is:

```
while [this condition is true]:
    [do this]
```

Here an example of a `while` loop:

```
password = input('Enter your password: ')
while password != 'unicOrn1990':
    print('The password is incorrect. Try again.')
    password = input('Enter your password: ')
print('The password is correct.')
```

This code keeps asking the user to enter the password while the password is incorrect.
Once the password is correct, the loop stops iterating.

DESCRIPTION

Create a script with a function `add_to_bookshelf()` that will add a book title to the bookshelf if there's an empty slot.

The function takes the arguments `book_to_add` (a string with a book title) and `bookshelf` (a list of book titles), and returns `True` if the bookshelf was updated, and `False` otherwise.

The function must use a `while` loop to iterate over the `bookshelf` list until it finds

an available slot (`'---'`). The function then replaces the `'---'` with the `book_to_add` and returns `True`. If there are no empty slots, the function doesn't change anything in the `bookshelf` and returns `False`.

The script in the EXAMPLE tests your function. If everything is correct, it should generate output as seen in the CONSOLE VIEW. Pay attention that you must only submit the file bookshelf.py, not the test script.

See the PYTHON INTERPRETER for more test cases.

EXAMPLE

```
# s01t05_while_main.py
from bookshelf import add_to_bookshelf


def print_bookshelf(shelf, updated):
    print(f'* Bookshelf. Updated: {updated} *', *shelf, '***\n', sep='\n')


if __name__ == '__main__':
    bookshelf = ['To Kill a Mockingbird', 'Little Women', '1984',
                 '---', 'Sense and Sensibility', '---', 'Lord of the Flies']
    print_bookshelf(bookshelf, False)

    was_updated = add_to_bookshelf('The Stranger', bookshelf)
    print_bookshelf(bookshelf, was_updated)

    was_updated = add_to_bookshelf('Dracula', bookshelf)
    print_bookshelf(bookshelf, was_updated)

    # adding a book to a full bookshelf
    was_updated = add_to_bookshelf('The Raven', bookshelf)
    print_bookshelf(bookshelf, was_updated)
```

CONSOLE VIEW

```
>python3 s01t05_while_main.py
* Bookshelf. Updated: False *
To Kill a Mockingbird
Little Women
1984
---
Sense and Sensibility
---
Lord of the Flies
***

* Bookshelf. Updated: True *
```

```
To Kill a Mockingbird
Little Women
1984
The Stranger
Sense and Sensibility
---
Lord of the Flies
***

* Bookshelf. Updated: True *
To Kill a Mockingbird
Little Women
1984
The Stranger
Sense and Sensibility
Dracula
Lord of the Flies
***

* Bookshelf. Updated: False *
To Kill a Mockingbird
Little Women
1984
The Stranger
Sense and Sensibility
Dracula
Lord of the Flies
***

>
```

PYTHON INTERPRETER

```
>python3
>>> from bookshelf import add_to_bookshelf
>>> my_bookshelf = ['---']
>>> add_to_bookshelf('It', my_bookshelf)
True
>>> my_bookshelf
['It']
>>> add_to_bookshelf('Beowulf', my_bookshelf)
False
>>> my_bookshelf
['It']
>>> my_bookshelf = ['It', '---', 'The Color Purple', '---']
>>> add_to_bookshelf('Beowulf', my_bookshelf)
True
>>> my_bookshelf
['It', 'Beowulf', 'The Color Purple', '---']
```

```
>>> my_bookshelf = []
>>> add_to_bookshelf('Beowulf', my_bookshelf)
False
>>> my_bookshelf
[]
>>>
```

SEE ALSO

Computer Programming - Loops
Python While Loop

Act Basic: Task 06

NAME

For

DIRECTORY

`t06_for/`

SUBMIT

`book_manager.py`

BEFORE YOU BEGIN

In this task you will also be working with loops, but, this time, with a `for` loop. Instead of specifying an exit condition, a `for` loop requires explicit instructions on what to iterate over. This works best for situations where you want to give instructions like: 'for every element do this', or 'repeat this action this many times'.

In order to complete this task, you must be able to answer the following questions:

- What is the syntax of a `for` loop in Python?
- How is a `for` loop different from `while`?
- How to iterate over the elements of a list or a string without using an index variable?
- How to iterate a variable over a given integer range? (e.g., from 5 to 1000)

DESCRIPTION

Create a script with a function `get_anonymous()` that will take a list of book titles `books` and return a new list containing the book titles that don't include an author.

The function must use a `for` loop to iterate over the `books` list. For every title, the function must check if it has the word `'by'` surrounded by at least one space before and after. If not, the title is appended to the new list.

The script in the EXAMPLE tests your function. If everything is correct, it should generate output as seen in the CONSOLE VIEW. Pay attention that you must only submit the file book_manager.py, not the test script.

See the PYTHON INTERPRETER for more test cases.

EXAMPLE

```
# s01t06_for_main.py
from book_manager import get_anonymous

if __name__ == '__main__':
    books = ['The Canterbury Tales by Geoffrey Chaucer',
             'A Separate Peace by John Knowles',
             'Mythology by Edith Hamilton',
```

```
            'Moby-Dick or, the Whale',
            'The Awakening by Kate Chopin',
            'Frankenstein',
            'Much Ado About Nothing by William Shakespeare',
            'Oliver Twist by Charles Dickens',
            'The Arabian Nights',
            'The Dream of the Red Chamber'
            ]
    print(*get_anonymous(books), '***', sep='\n')
    # test cases for error management
    print(*get_anonymous([
        'The Crucible by Arthur Miller',
        'The Crucible byby Arthur Miller',  # error: no space after 'by'
        'The Crucible by by Arthur Miller',
        'The Crucible byArthur Miller',  # error: no space after 'by'
        'The Crucibleby Arthur Miller',  # error: no space before 'by'
        'The Crucible     by     Arthur Miller',
        'The Crucible by Arthur Miller by Arthur Miller',
        'by Arthur Miller The Crucible',  # error: no space before 'by'
        ' by Arthur Miller The Crucible',
        ' by '
    ]), sep='\n')
```

CONSOLE VIEW

```
>python3 s01t06_for_main.py
Moby-Dick or, the Whale
Frankenstein
The Arabian Nights
The Dream of the Red Chamber
***
The Crucible byby Arthur Miller
The Crucible byArthur Miller
The Crucibleby Arthur Miller
by Arthur Miller The Crucible
>
```

PYTHON INTERPRETER

```
>python3
>>> from book_manager import get_anonymous
>>> empty_book_list = []
>>> get_anonymous(empty_book_list)
[]
>>> books = ['Macbeth']
>>> get_anonymous(books)
['Macbeth']
```

```
>>> books = ['Macbeth', 'Animal Farm by George Orwell']
>>> get_anonymous(books)
['Macbeth']
>>> books = ['Animal Farm by George Orwell']
>>> get_anonymous(books)
[]
>>>
```

SEE ALSO

For Loops

Act Basic: Task 07

NAME

Analytics

DIRECTORY

`t07_analytics/`

SUBMIT

`analytics.py`

BEFORE YOU BEGIN

In order to complete this task, you must be able to answer the following questions:

- What is a character in programming?
- What characteristics and types can characters have?
- How to analyze a string for these characteristics (what string methods in Python can help with this)?

DESCRIPTION

Create a script that contains a function `print_str_analytics()`.
It must take one string as a parameter and print information about it.
The information must include the number of:

- printable characters
- alphanumeric characters
- alphabetic characters
- decimal characters
- lowercase letters
- uppercase letters
- whitespace characters

The script in the EXAMPLE tests your function. Use this as an example, add your own values, and also test your function as shown in the PYTHON INTERPRETER. If everything is correct, it should generate output as shown below. Pay close attention to the formatting of the output. Pay attention that you must only submit the file analytics.py, not the test script, and Oracle will check your function with random values.

EXAMPLE

```
# s01t07_analytics_main.py

from analytics import print_str_analytics


if __name__ == '__main__':
    print_str_analytics('We are three wise men!')
    print_str_analytics('NI')
```

CONSOLE VIEW

```
>python3 s01t07_analytics_main.py
|-----------------------------------------------|
|               String analytics                |
|-----------------------------------------------|
| 'We are three wise men!'
|-----------------------------------------------|
| Number of printable characters is: 22
| Number of alphanumeric characters is: 17
| Number of alphabetic characters is: 17
| Number of decimal characters is: 0
| Number of lowercase letters is: 16
| Number of uppercase letters is: 1
| Number of whitespace characters is: 4
|-----------------------------------------------|
|-----------------------------------------------|
|               String analytics                |
|-----------------------------------------------|
| 'NI'
|-----------------------------------------------|
| Number of printable characters is: 2
| Number of alphanumeric characters is: 2
| Number of alphabetic characters is: 2
| Number of decimal characters is: 0
| Number of lowercase letters is: 0
| Number of uppercase letters is: 2
| Number of whitespace characters is: 0
|-----------------------------------------------|
>
```

PYTHON INTERPRETER

```
>python3
>>> import analytics
>>> analytics.print_str_analytics('16 years behind a veil and proud of it, sir.')
|-----------------------------------------------|
|               String analytics                |
|-----------------------------------------------|
| '16 years behind a veil and proud of it, sir.'
|-----------------------------------------------|
| Number of printable characters is: 44
| Number of alphanumeric characters is: 33
| Number of alphabetic characters is: 31
| Number of decimal characters is: 2
| Number of lowercase letters is: 31
| Number of uppercase letters is: 0
| Number of whitespace characters is: 9
|-----------------------------------------------|
>>>
```

Act Advanced: Task 08

NAME

Validator

DIRECTORY

`t08_validator/`

SUBMIT

`validator.py`

DESCRIPTION

Create a script that contains a function called `validator()`.
The function:

- takes an argument in the following format:
  `'number_1 operator number_2'`
- casts each of the numbers into `float`
- if the casting is impossible, returns `False`
- if the operator is invalid, returns `False`
- if the expression is true, returns `True`
- if the expression is incorrect, corrects the operator, and returns the corrected expression as a string

The operator may be: `>`, `<`, `>=`, `<=`, or `==`. The operator correction works as follows:

- `>` replaces `<`, and vice versa
- `>=` replaces `<=`, and vice versa
- `>=` or `<=` replace `==`

Test your function as shown in the PYTHON INTERPRETER.

PYTHON INTERPRETER

```
>python3
>>> from validator import validator
>>> validator('4 < 6')
True
>>> validator('4 > 6')
'4.0 < 6.0'
>>> validator('4 > ')
False
>>> validator('4  6')
False
>>> validator('4 . 6')
False
>>> validator('4 == 6')
'4.0 <= 6.0'
```

```
>>> validator('4 >= 6')
'4.0 <= 6.0'
>>>
```

Act Advanced: Task 09

NAME

Byte me

DIRECTORY

`t09_byte_me/`

SUBMIT

`bytes.py`

DESCRIPTION

Create a script that contains a function called `convert_to_bytes()`.
The function:

- takes three positional arguments as strings
- casts the first argument to integer, the second to boolean (True, False) and the third to string
- converts these variables into `bytes`
- prints the values and their bytes representations in the following format:
  '-- The [data type] value is "[original value]"
  bytes: "[byte representation]"'
- does not handle invalid values

PYTHON INTERPRETER

```
>python3
>>> from bytes import convert_to_bytes
>>> convert_to_bytes('10', 'False', 'aaa')
-- The int value is "10"
bytes: "b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'"
-- The bool value is "False"
bytes: "b''"
-- The string value is "aaa"
bytes: "b'aaa'"
>>> convert_to_bytes('5', 'True', 'Are you suggesting that coconuts migrate?')
-- The int value is "5"
bytes: "b'\x00\x00\x00\x00\x00'"
-- The bool value is "True"
bytes: "b'\x00'"
-- The string value is "Are you suggesting that coconuts migrate?"
bytes: "b'Are you suggesting that coconuts migrate?'"
>>> convert_to_bytes('1', 'true', 'bbb')
-- The int value is "1"
bytes: "b'\x00'"
-- The bool value is "False"
bytes: "b''"
-- The string value is "bbb"
```

```
bytes: "b'bbb'"
>>>
```

Act Advanced: Task 10

NAME

Top secret

DIRECTORY

`t10_top_secret/`

SUBMIT

`hash.py`

BEFORE YOU BEGIN

In order to complete this task, you must be able to answer the following questions:

- What is hashing in programming? What is the purpose of hashing?
- What is hash value of a string?
- Is it possible to hash a string in Python? What features can help?

DESCRIPTION

Create a script that contains two functions, each of which encrypts a message given as an argument using one of the algorithms (listed below), and prints the result.

The algorithms must be `md5` and `sha1`.
The respective functions are `md5_hash` and `sha1_hash`.
In your script, encrypt a string with each of the functions, and print the result as shown in the PYTHON INTERPRETER.

PYTHON INTERPRETER

```
>python3
>>> from hash import md5_hash, sha1_hash
>>> md5_hash('My hovercraft is full of eels')
Original string: My hovercraft is full of eels
md5 hash generated is
01af419b4ccbc18ea6ad64422ba94d34
>>> sha1_hash('My hovercraft is full of eels')
Original string: My hovercraft is full of eels
sha1 hash generated is
10f0f962dcfbf9e5765181055aa11a5f4a3f6ecf
>>>
```

SEE ALSO

Hashing Strings with Python
hashlib

Share

PUBLISHING

Last but not least, the final stage of your work is to publish it. This allows you to share your challenges, solutions, and reflections with local and global audiences. During this stage, you will discover ways of getting external evaluation and feedback on your work. As a result, you will get the most out of the challenge, and get a better understanding of both your achievements and missteps.

To share your work, you can create:

- a text post, as a summary of your reflection
- charts, infographics or other ways to visualize your information
- a video, either of your work, or a reflection video
- an audio podcast. Record a story about your experience
- a photo report with a small post

Helpful tools:

- Canva - a good way to visualize your data
- QuickTime - an easy way to capture your screen, record video or audio

Examples of ways to share your experience:

- Facebook - create and share a post that will inspire your friends
- YouTube - upload an exciting video
- GitHub - share and describe your solution
- Telegraph - create a post that you can easily share on Telegram
- Instagram - share photos and stories from ucode. Don't forget to tag us :)

Share what you've learned and accomplished with your local community and the world. Use #ucode and #CBLWorld on social media.