

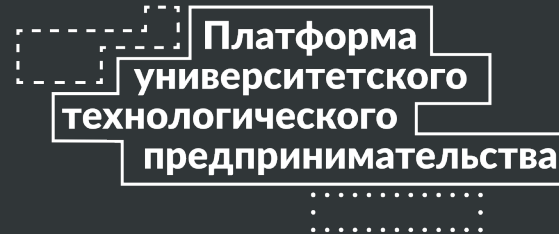


Мастер-класс



Программные средства
искусственного интеллекта

17.10.2022



Где применяется ИИ?

Стартап вопросы:

- **Вопрос о технологии:** Способны ли вы создать продвинутую технологию, а не вносить мелкие дополнения в уже существующую?

Стартап вопросы:

- **Вопрос о технологии:** Способны ли вы создать продвинутую технологию, а не вносить мелкие дополнения в уже существующую?
- **Вопрос о времени:** Подходящий ли сейчас момент начинать задуманный вами бизнес?

Стартап вопросы:

- **Вопрос о технологии:** Способны ли вы создать продвинутую технологию, а не вносить мелкие дополнения в уже существующую?
- **Вопрос о времени:** Подходящий ли сейчас момент начинать задуманный вами бизнес?
- **Вопрос о монополии:** Вы начинаете с захвата большой доли на маленьком рынке?

Стартап вопросы:

- **Вопрос о технологии:** Способны ли вы создать продвинутую технологию, а не вносить мелкие дополнения в уже существующую?
- **Вопрос о времени:** Подходящий ли сейчас момент начинать задуманный вами бизнес?
- **Вопрос о монополии:** Вы начинаете с захвата большой доли на маленьком рынке?
- **Вопрос о людях:** У вас достойная команда?

Стартап вопросы:

- **Вопрос о технологии:** Способны ли вы создать продвинутую технологию, а не вносить мелкие дополнения в уже существующую?
- **Вопрос о времени:** Подходящий ли сейчас момент начинать задуманный вами бизнес?
- **Вопрос о монополии:** Вы начинаете с захвата большой доли на маленьком рынке?
- **Вопрос о людях:** У вас достойная команда?
- **Вопрос о продажах:** У вас есть возможность не только создать, но и продавать ваш продукт?

Стартап вопросы:

- **Вопрос о технологии:** Способны ли вы создать продвинутую технологию, а не вносить мелкие дополнения в уже существующую?
- **Вопрос о времени:** Подходящий ли сейчас момент начинать задуманный вами бизнес?
- **Вопрос о монополии:** Вы начинаете с захвата большой доли на маленьком рынке?
- **Вопрос о людях:** У вас достойная команда?
- **Вопрос о продажах:** У вас есть возможность не только создать, но и продавать ваш продукт?
- **Вопрос о времени жизни:** Сможете ли вы сохранить свои позиции на рынке через 10 лет? А через 20?

Стартап вопросы:

- **Вопрос о технологии:** Способны ли вы создать продвинутую технологию, а не вносить мелкие дополнения в уже существующую?
- **Вопрос о времени:** Подходящий ли сейчас момент начинать задуманный вами бизнес?
- **Вопрос о монополии:** Вы начинаете с захвата большой доли на маленьком рынке?
- **Вопрос о людях:** У вас достойная команда?
- **Вопрос о продажах:** У вас есть возможность не только создать, но и продавать ваш продукт?
- **Вопрос о времени жизни:** Сможете ли вы сохранить свои позиции на рынке через 10 лет? А через 20?
- **Вопрос об открытии:** Нашли ли вы свой уникальный шанс, не замеченный остальными?





```
# Что делает код?
```

```
t = [-5, -10, 1, 11, 20, 25, 27, 23, 18, 8, 2, -3]
```

```
s = 0
```

```
mm = 1000
```

```
mx = -1000
```

```
for e in t:
```

```
    s += e
```

```
    if e < mm:
```

```
        mm = e
```

```
    if e > mx:
```

```
        mx = e
```

```
print(s / len(t))
```

```
print(mm)
```

```
print(mx)
```





```
# Та же программа
```

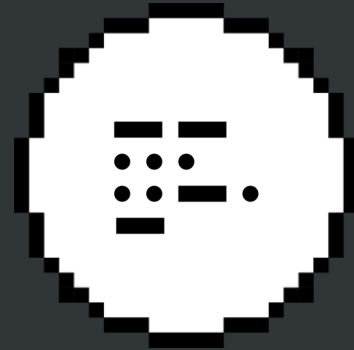
```
temperatures = [-5, -10, 1, 11, 20, 25, 27, 23, 18, 8, 2, -3]
average_temperature = sum(temperatures) / len(temperatures)
print(average_temperature)
print(min(temperatures))
print(max(temperatures))
```



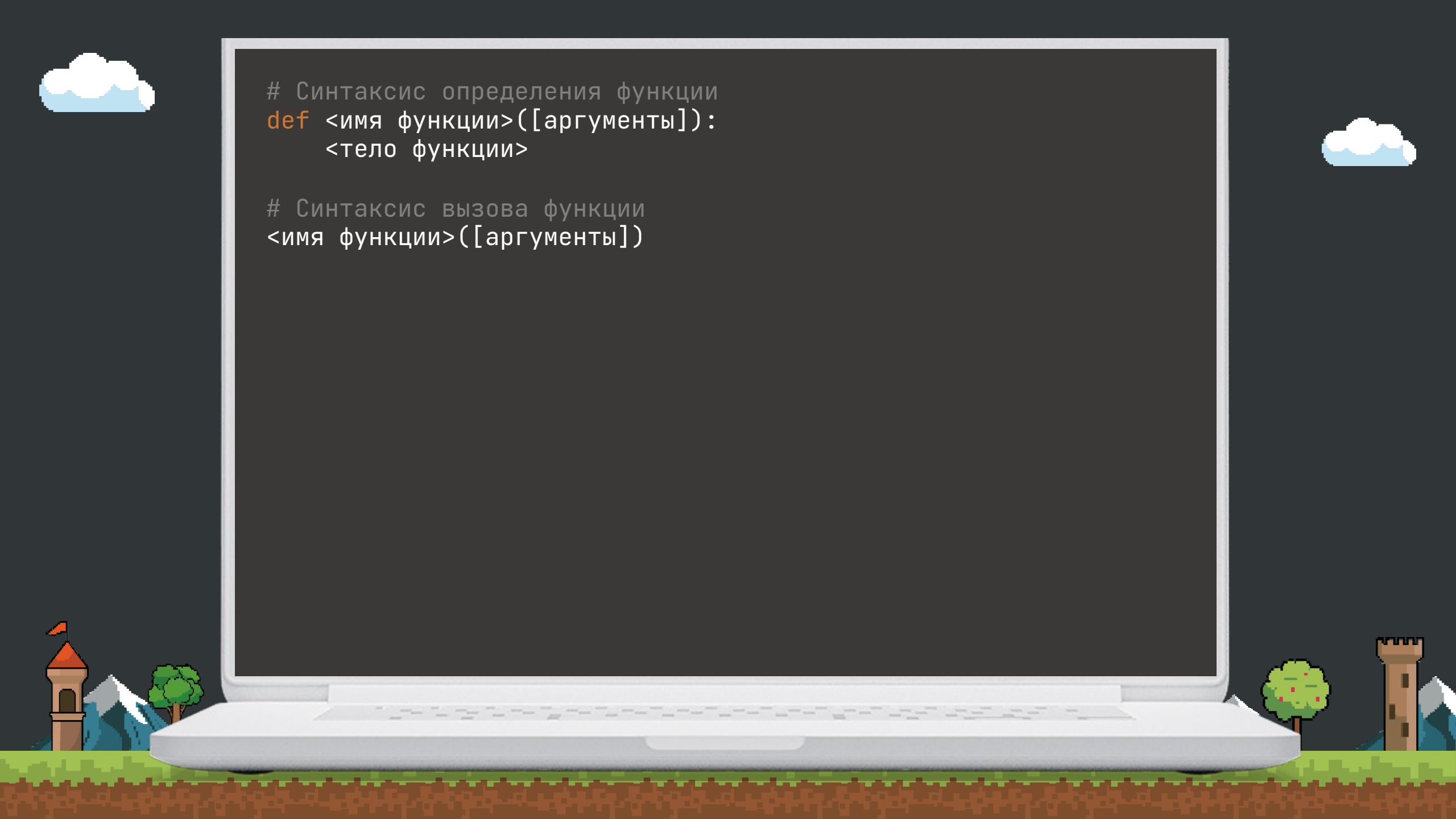
`for (loop)`



`while (loop)`

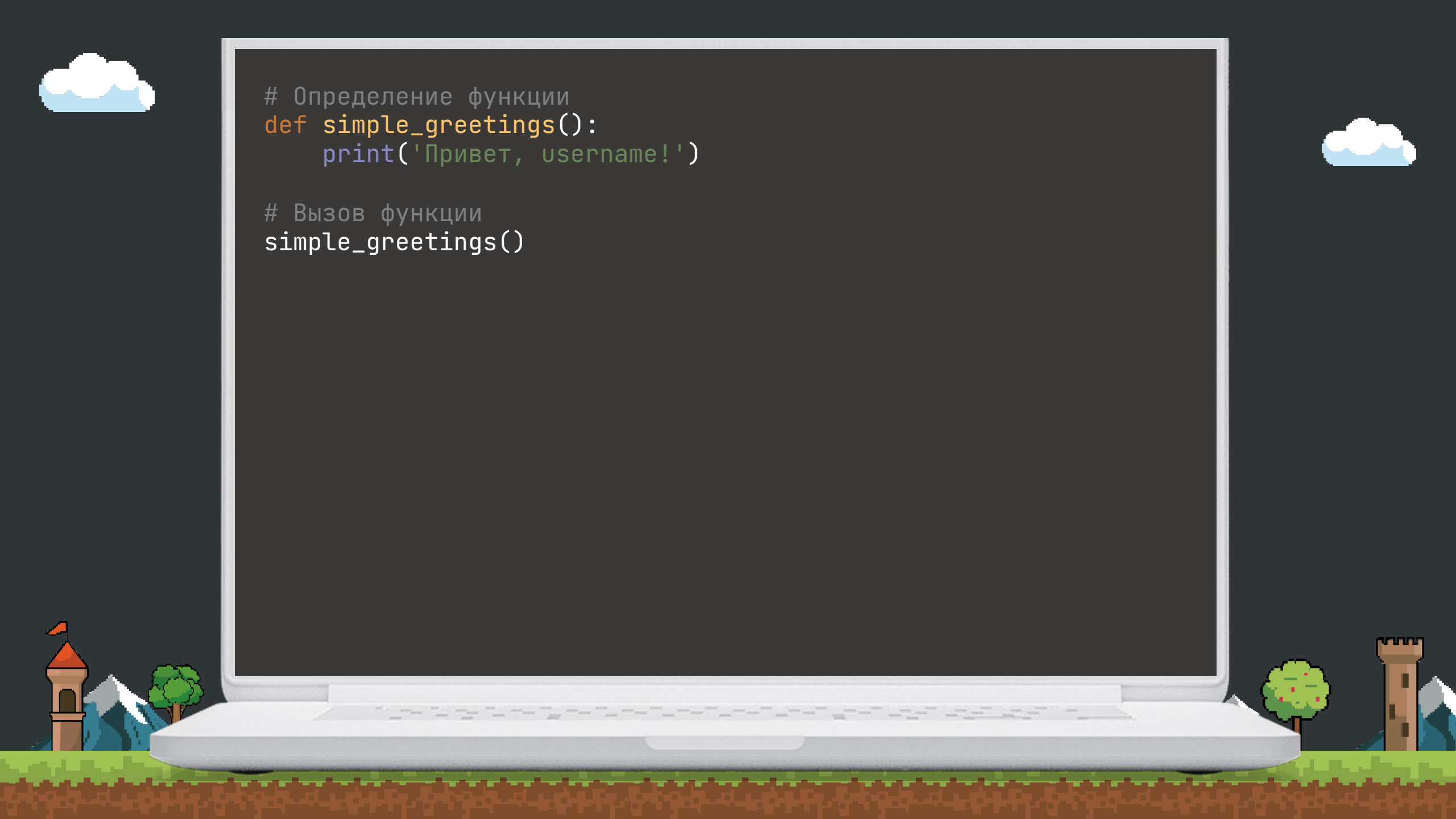


`function`



```
# Синтаксис определения функции  
def <имя функции>([аргументы]):  
    <тело функции>
```

```
# Синтаксис вызова функции  
<имя функции>([аргументы])
```



```
# Определение функции
def simple_greetings():
    print('Привет, username!')

# Вызов функции
simple_greetings()
```



```
# Порядок вызова имеет значение
```

```
simple_greetings2()
```

```
def simple_greetings2():  
    print('Привет, username!')
```

```
Traceback (most recent call last):  
  File "/Users/levbrave/test.py", line 3, in <module>  
    simple_greetings2()  
NameError: name 'simple_greetings2' is not defined
```



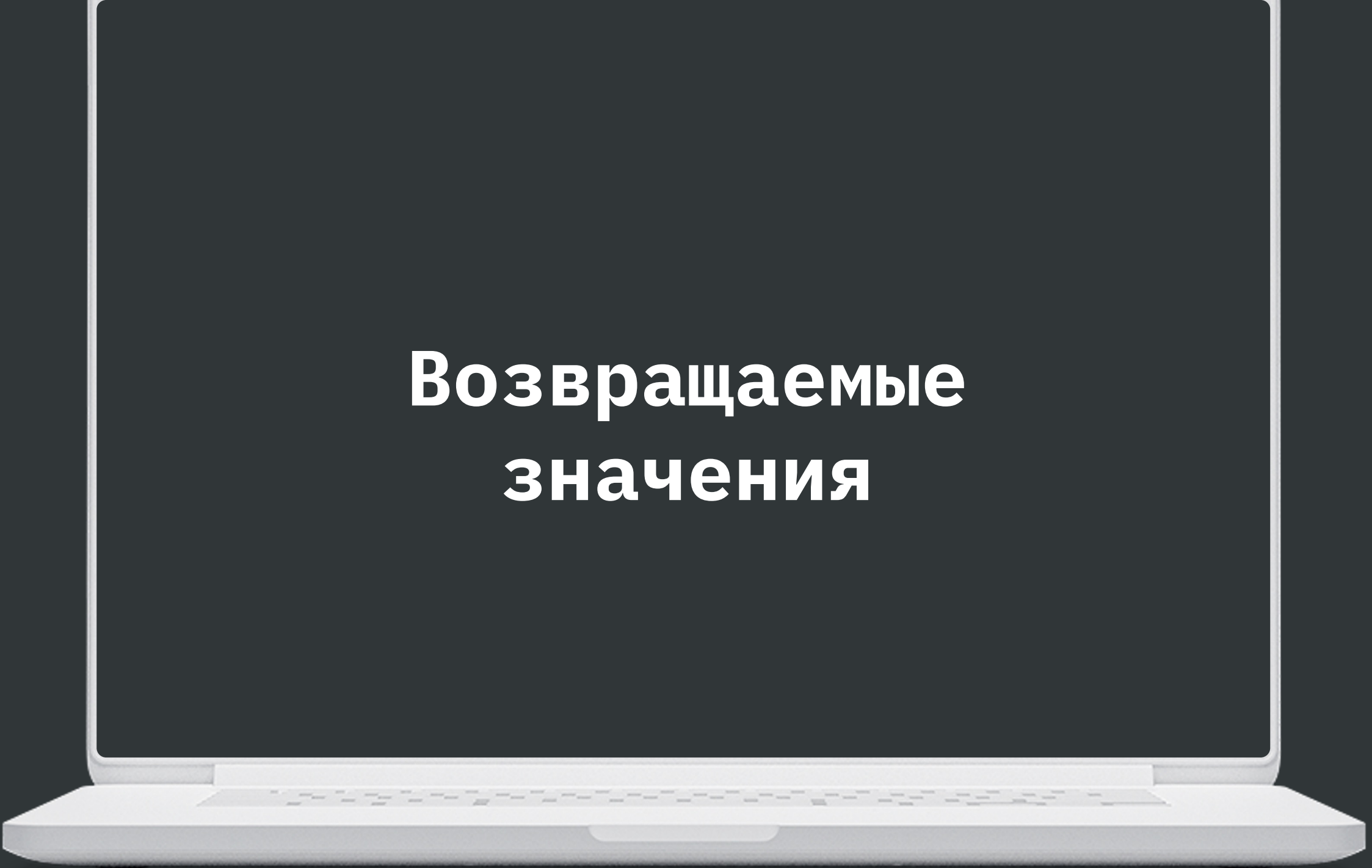

```
# Использование аргументов
```

```
def print_array(array):  
    for element in array:  
        print(element)
```

```
print_array(['Hello', 'world'])  
print()  
print_array([123, 456, 789])
```

```
Hello  
world
```

```
123  
456  
789
```

A white laptop is shown from a front-facing perspective, slightly angled. The screen is dark and displays the text 'Возвращаемые значения' in a bold, white, sans-serif font. The laptop's keyboard and trackpad are visible at the bottom of the frame.

**Возвращаемые
значения**

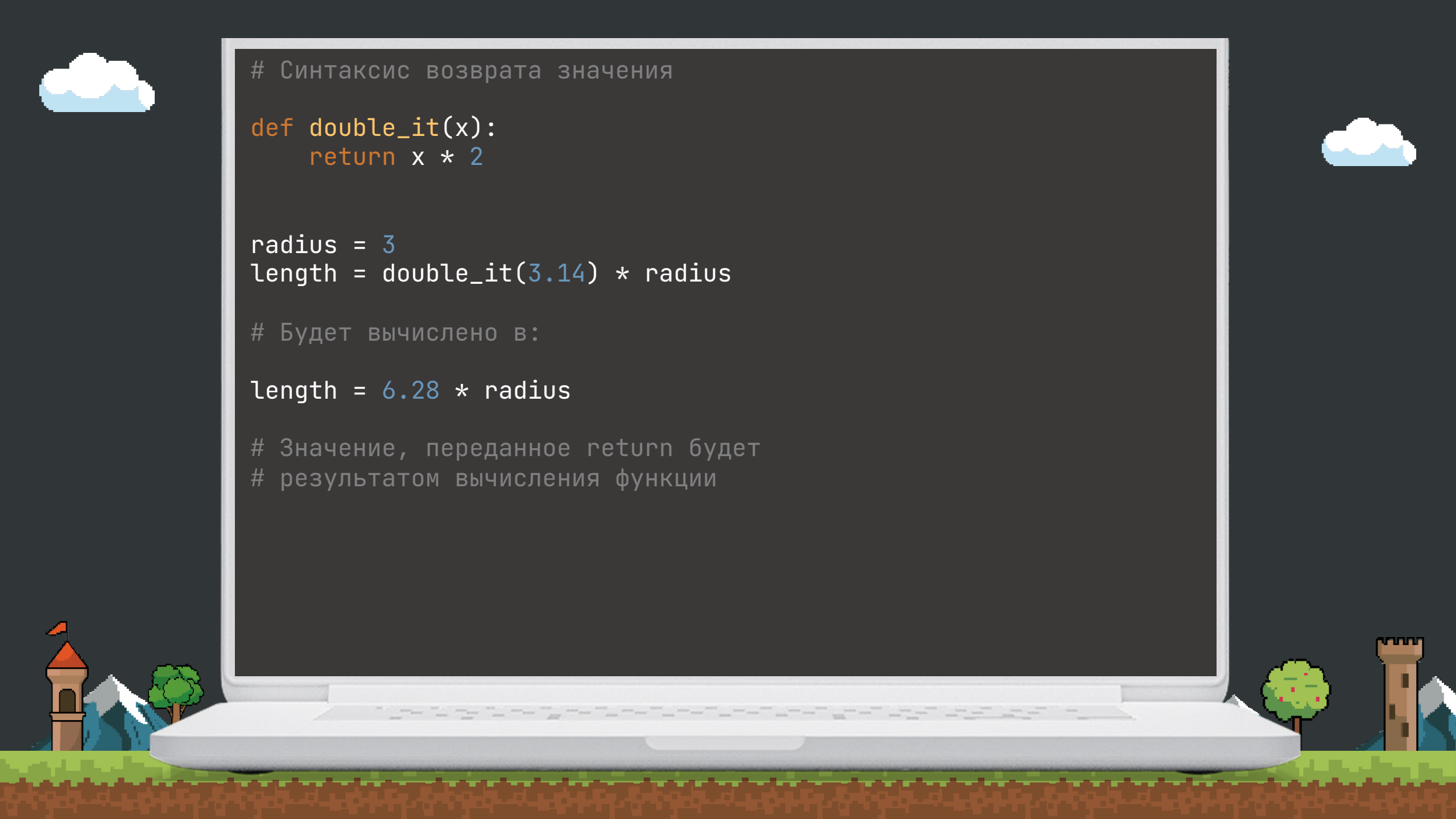


```
# Синтаксис возврата значения
```

```
def double_it(x):  
    return x * 2
```

```
radius = 3  
length = double_it(3.14) * radius
```

```
# Как только выполнение доходит до инструкции return,  
# выполнение функции завершается и интерпретатор  
# возвращается к месту, где функция была вызвана
```



```
# Синтаксис возврата значения
```

```
def double_it(x):  
    return x * 2
```

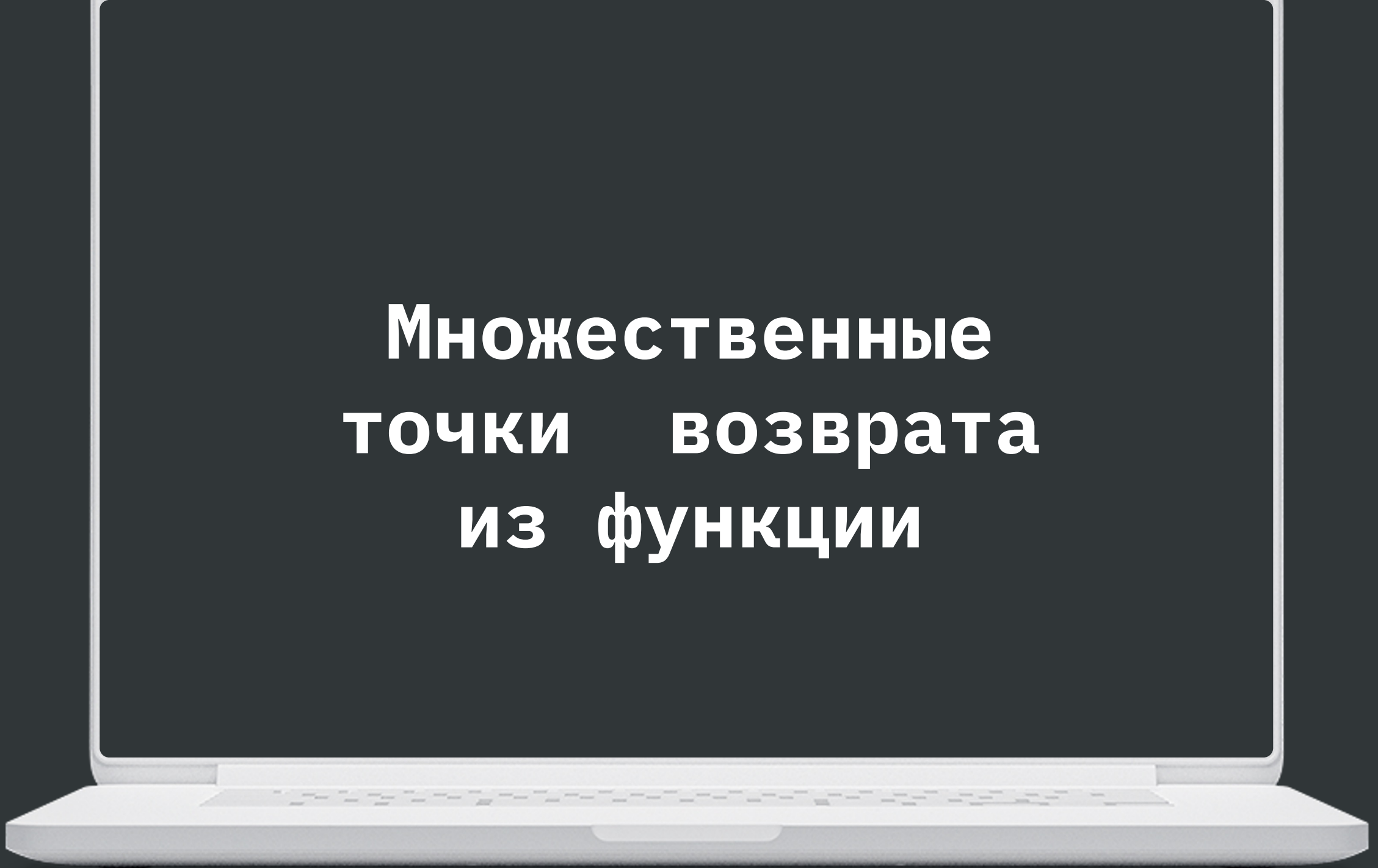
```
radius = 3  
length = double_it(3.14) * radius
```

```
# Будет вычислено в:
```

```
length = 6.28 * radius
```

```
# Значение, переданное return будет  
# результатом вычисления функции
```

**Множественные
точки возврата
из функции**





```
# Несколько ветвей вычисления
```

```
def my_abs(x):  
    if x >= 0:  
        result = x  
    else:  
        result = -x  
  
    return result
```

```
# Упростим функцию
```



```
# Несколько точек возврата
```

```
def my_abs(x):  
    if x >= 0:  
        result = x  
        return result  
    else:  
        result = -x  
        return result
```

```
# Упростим функцию
```



```
# Несколько точек возврата
```

```
def my_abs(x):  
    if x >= 0:  
        return x  
    else:  
        return -x
```


A white laptop is shown from a front-facing perspective, slightly angled. The screen is dark and displays the text "Возврат из глубины функции" in white. The laptop's keyboard and trackpad are visible at the bottom.

**Возврат из
глубины функции**



```
# Выход из нескольких уровней вложенности
```

```
def matrix_has_close_value(matrix, value, eps):  
    found = False  
    for row in matrix:  
        for cell in row:  
            if abs(cell - value) <= eps:  
                found = True  
                break  
        if found:  
            break  
    if found:  
        return True  
    else:  
        return False
```

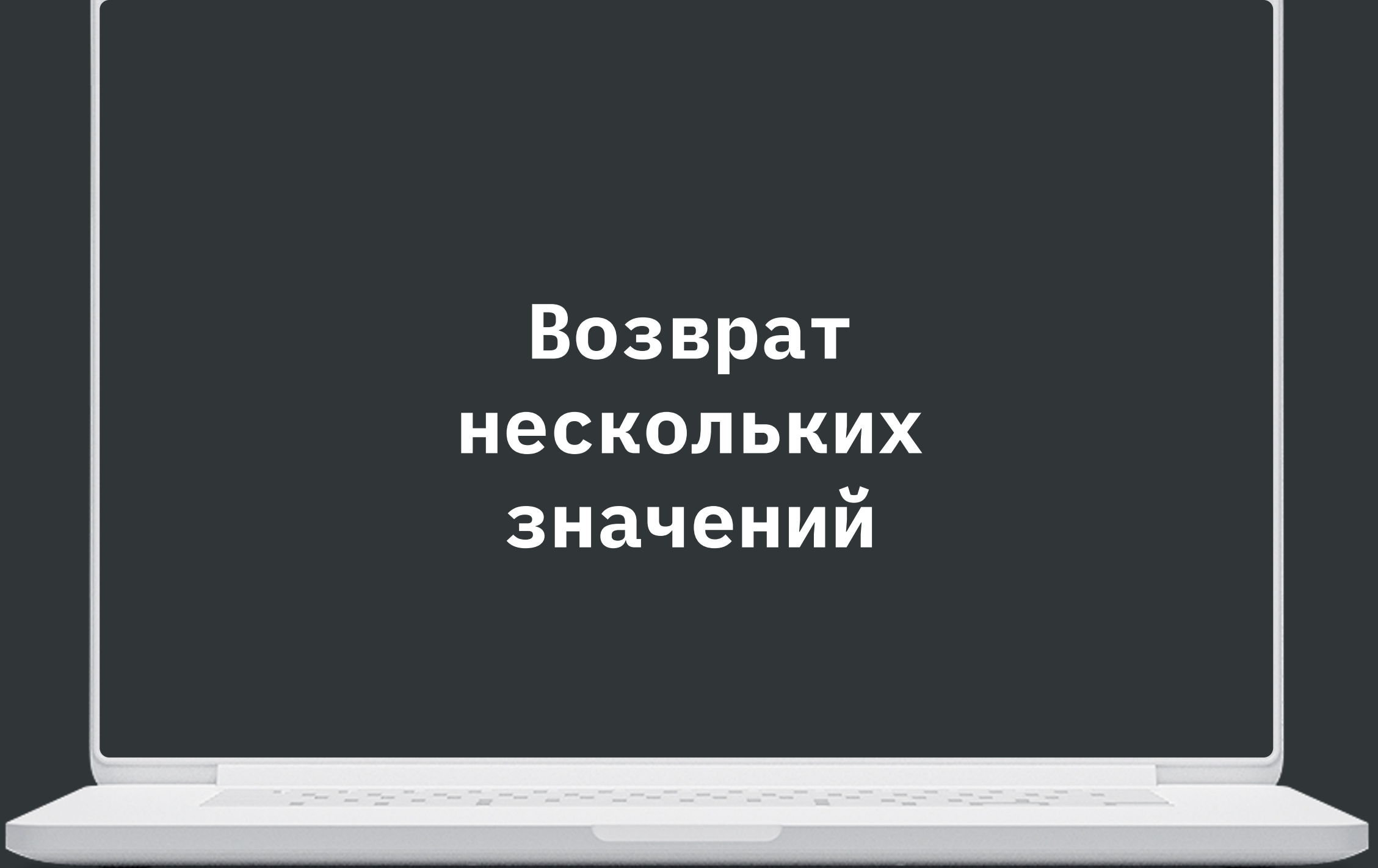


```
# Выход из нескольких уровней вложенности
```

```
def matrix_has_close_value(matrix, value, eps):  
    for row in matrix:  
        for cell in row:  
            if abs(cell - value) <= eps:  
                return True  
    return False
```

```
# Оператор return позволяет выйти с любого уровня  
# вложенности функции
```

**Возврат
нескольких
значений**





```
# Возврат нескольких значений
```

```
def get_coordinates():  
    return 1, 2
```



```
print(get_coordinates()) # => (1, 2)
```

```
# Команда возврата нескольких значений
```

```
return 1, 2
```

```
# Практически идентична команде возврата кортежа  
# с этими значениями
```

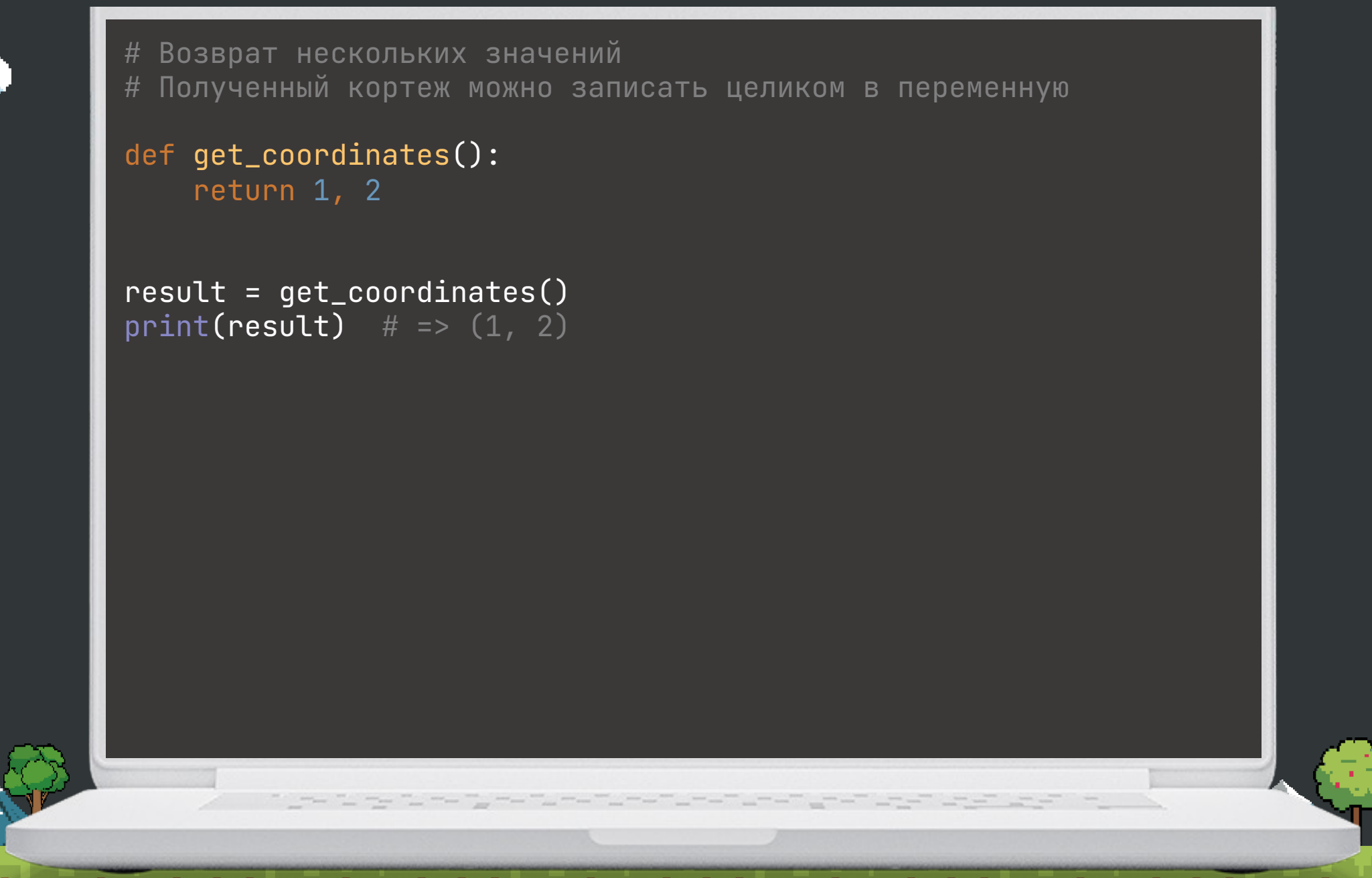
```
return (1, 2)
```



```
# Возврат нескольких значений
# Полученный кортеж можно записать целиком в переменную
```

```
def get_coordinates():
    return 1, 2
```

```
result = get_coordinates()
print(result)  # => (1, 2)
```





```
# Локальные переменные
```

```
def print_array(array):  
    for element in array:  
        print(element)
```

```
print_array(['Hello', 'world'])  
print_array([123, 456, 789])
```

```
# Переменные array и element – локальные  
# Они не существуют снаружи от функции
```



```
# Несколько имён одной переменной
```

```
def print_array(array):  
    for element in array:  
        print(element)
```

```
words = ['Hello', 'world']  
print_array(words)
```

```
# Обращаемся к массиву  
# по имени аргумента –  
# локальная переменная array
```

```
def print_array(array):  
    for element in words:  
        print(element)
```

```
words = ['Hello', 'world']  
print_array(words)
```

```
# Обращаемся к массиву  
# по имени внешней переменной –  
# глобальная переменная words
```




```
# Глобальные переменные плохи для передачи значения в функцию
```

```
def print_array(array):  
    for element in array:  
        print(element)
```

```
words = ['Hello', 'world']  
print_array(['abc', 'def'])
```

```
abc  
def
```

```
def print_array(array):  
    for element in words:  
        print(element)
```

```
words = ['Hello', 'world']  
print_array(['abc', 'def'])
```

```
Hello  
world
```



```
# Глобальные переменные - «константы»
```

```
ENGLISH_RAINBOW_COLORS = ['red', 'orange', 'yellow', 'green',  
                           'blue', 'indigo', 'violet']
```

```
RUSSIAN_RAINBOW_COLORS = ['красный', 'оранжевый', 'желтый',  
                          'зеленый', 'голубой', 'синий',  
                          'фиолетовый']
```

```
def rainbow_color(index, russian_or_english):  
    if russian_or_english == 'russian':  
        print(RUSSIAN_RAINBOW_COLORS[index])  
    elif russian_or_english == 'english':  
        print(ENGLISH_RAINBOW_COLORS[index])  
    else:  
        print('Неверный язык')
```

```
rainbow_color(2, 'russian')  
rainbow_color(2, 'english')
```



```
# Запись внешних переменных
```

```
area = 'Красная площадь'
```

```
def print_square_area(length, width):  
    area = length * width  
    print('Площадь площади: ', area)
```

```
print('Место встречи: ', area)  
print_square_area(330, 75)  
print('Повторяю, место встречи: ', area)
```



```
# Использование main
```

```
def print_square_area(length, width):  
    area = length * width  
    print('Площадь площади: ', area)
```

```
def main():  
    area = 'Красная площадь'  
    print('Место встречи: ', area)  
    print_square_area(330, 75)  
    print('Повторяю, место встречи: ', area)
```

```
main()
```



```
# Присваивание глобальных переменных
```

```
ask_number = 0
```

```
def ask_again():  
    global ask_number  
    ask_number = ask_number + 1  
    print('Ты спрашиваешь меня уже в', ask_number, '-й раз')
```

```
ask_again()  
ask_again()  
ask_again()
```



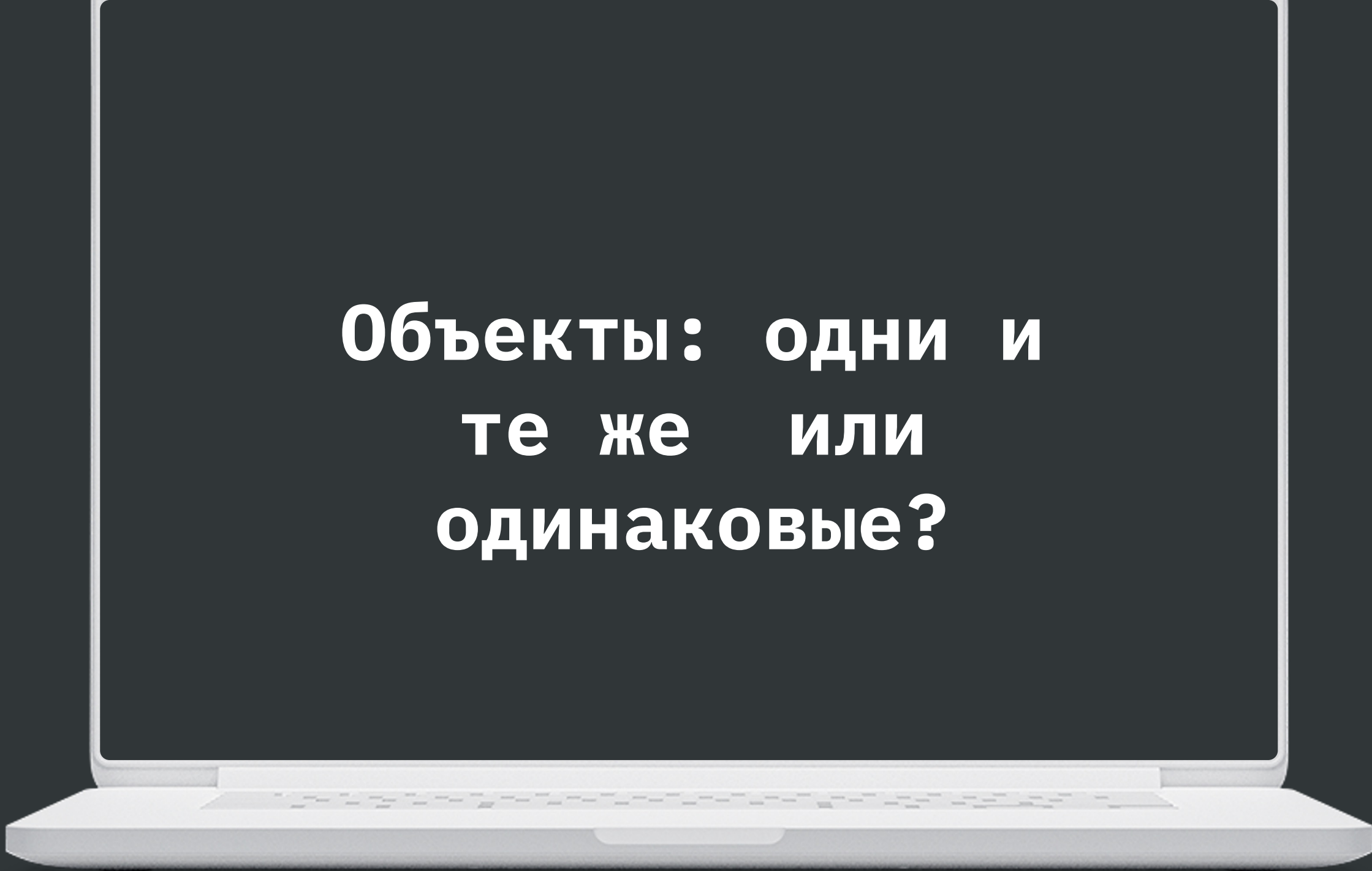
```
# Аргумент функции используется так же как локальная переменная
```

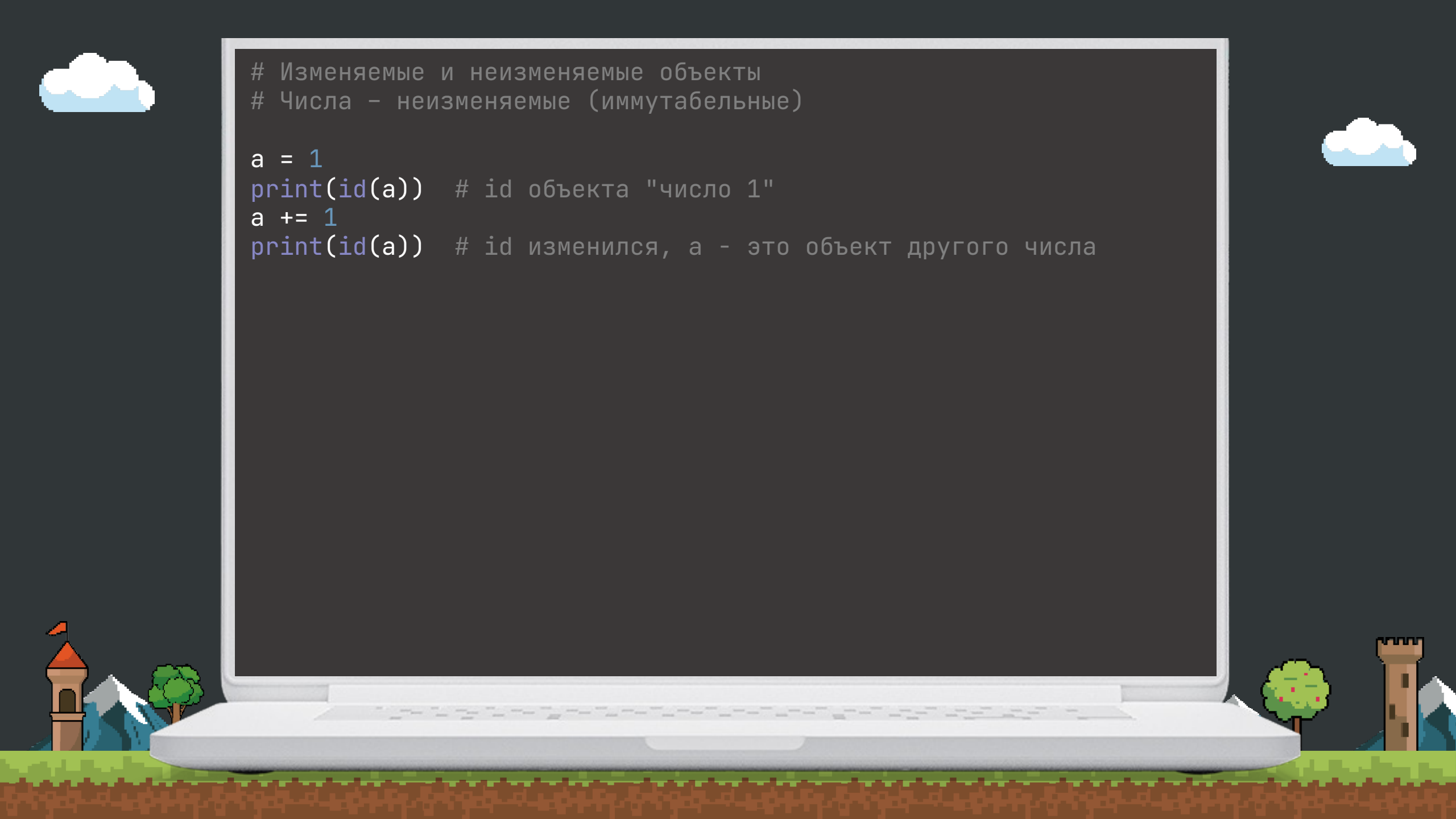
```
def greet(name):  
    print('Привет, ', name)  
    name = 'товарищ'  
    print('Здравствуй, ', name)
```

```
greet('Вася')
```

```
Привет, Вася  
Здравствуй, товарищ
```

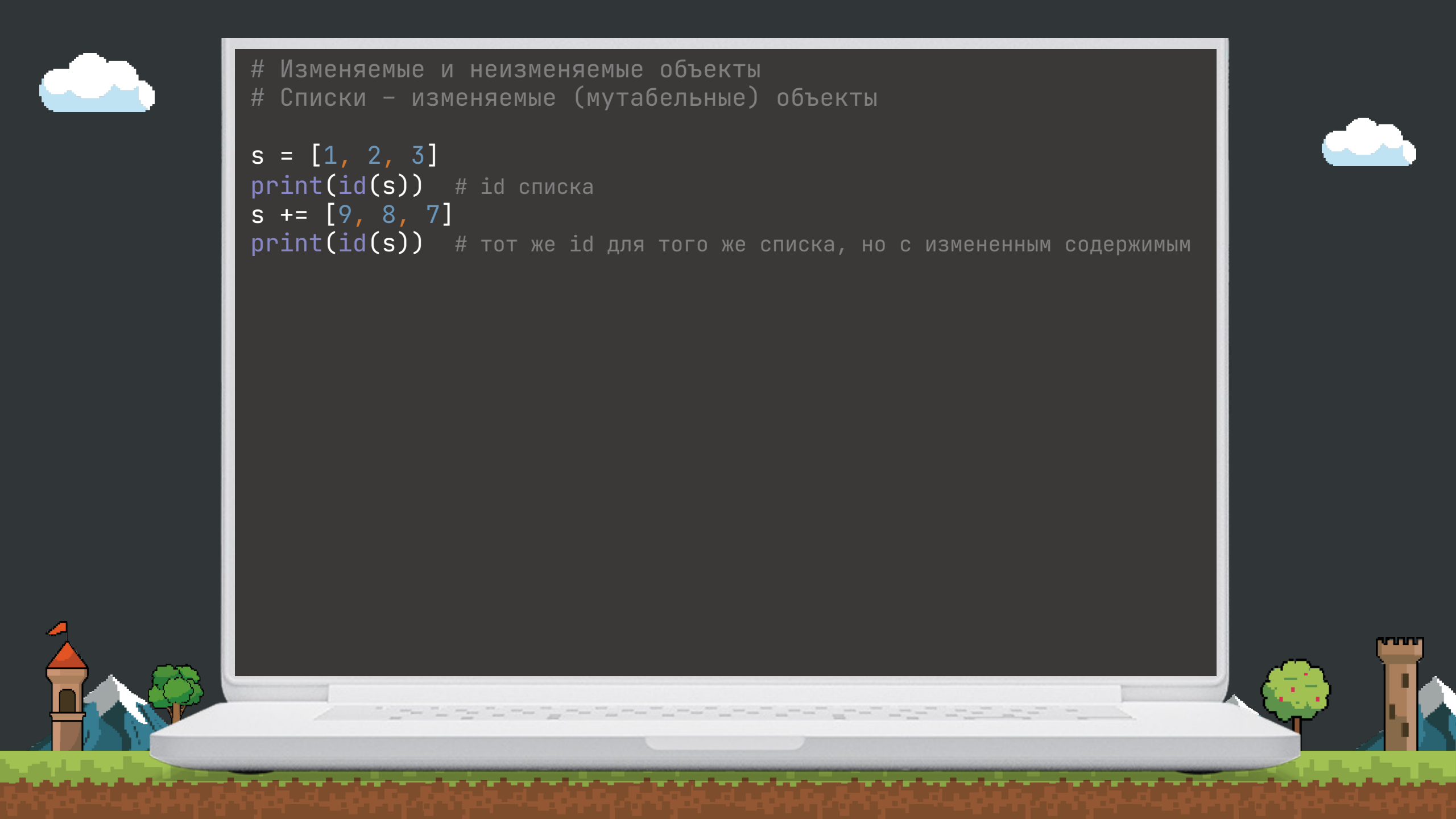
**Объекты: одни и
те же или
одинаковые?**





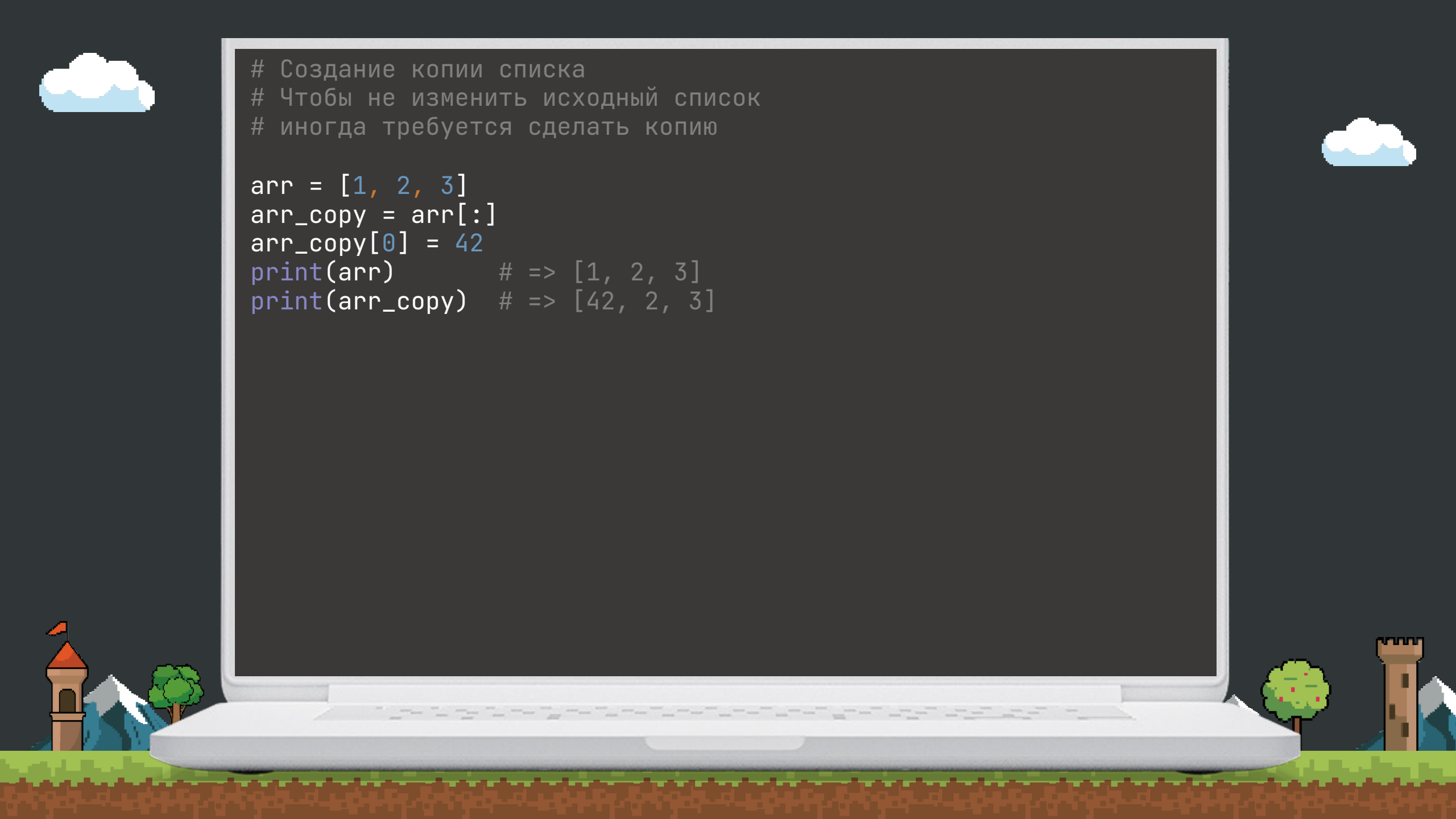
```
# Изменяемые и неизменяемые объекты
# Числа – неизменяемые (иммутабельные)

a = 1
print(id(a)) # id объекта "число 1"
a += 1
print(id(a)) # id изменился, а - это объект другого числа
```

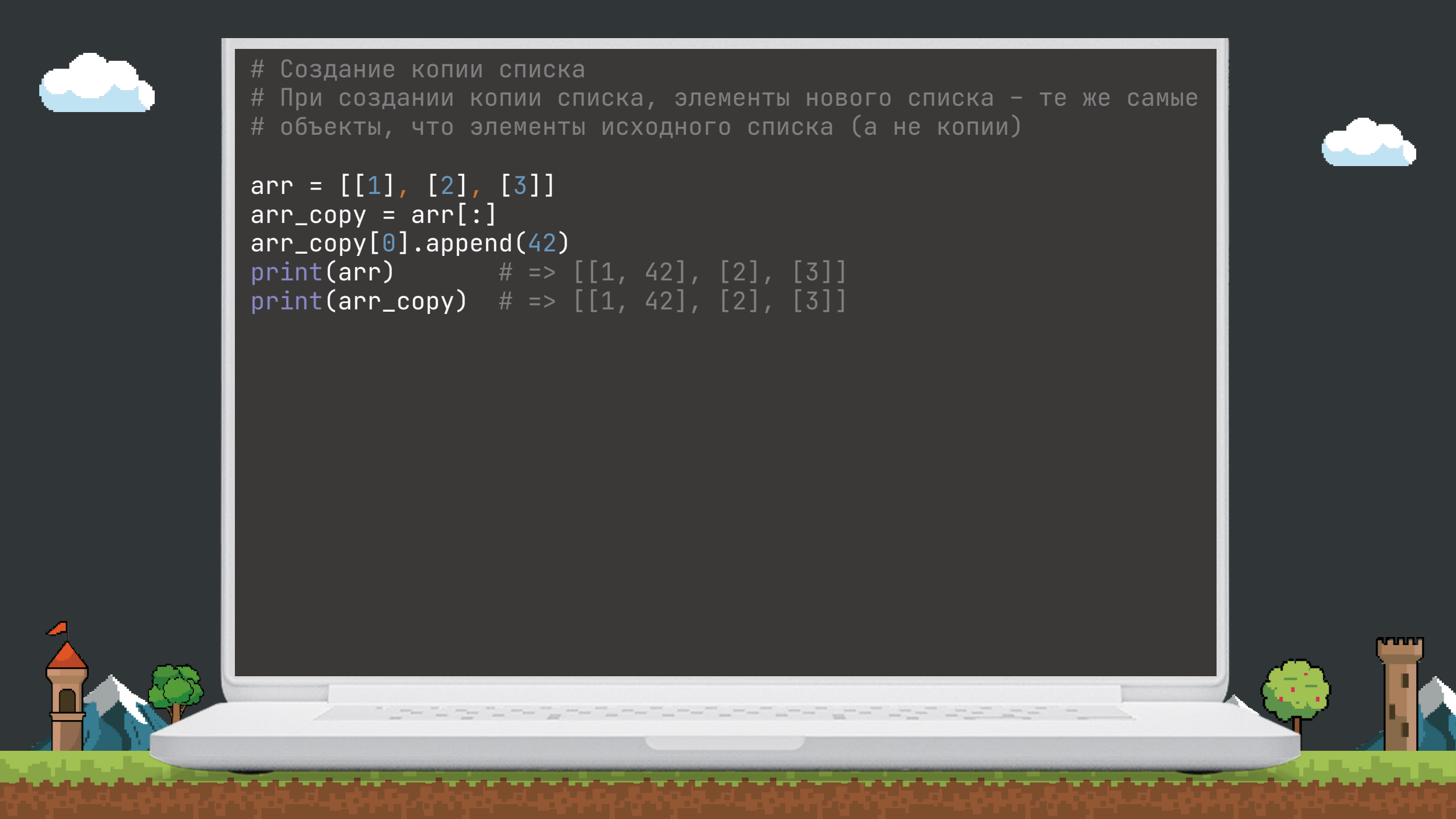
```
# Изменяемые и неизменяемые объекты
# Списки – изменяемые (мутабельные) объекты

s = [1, 2, 3]
print(id(s)) # id списка
s += [9, 8, 7]
print(id(s)) # тот же id для того же списка, но с измененным содержимым
```



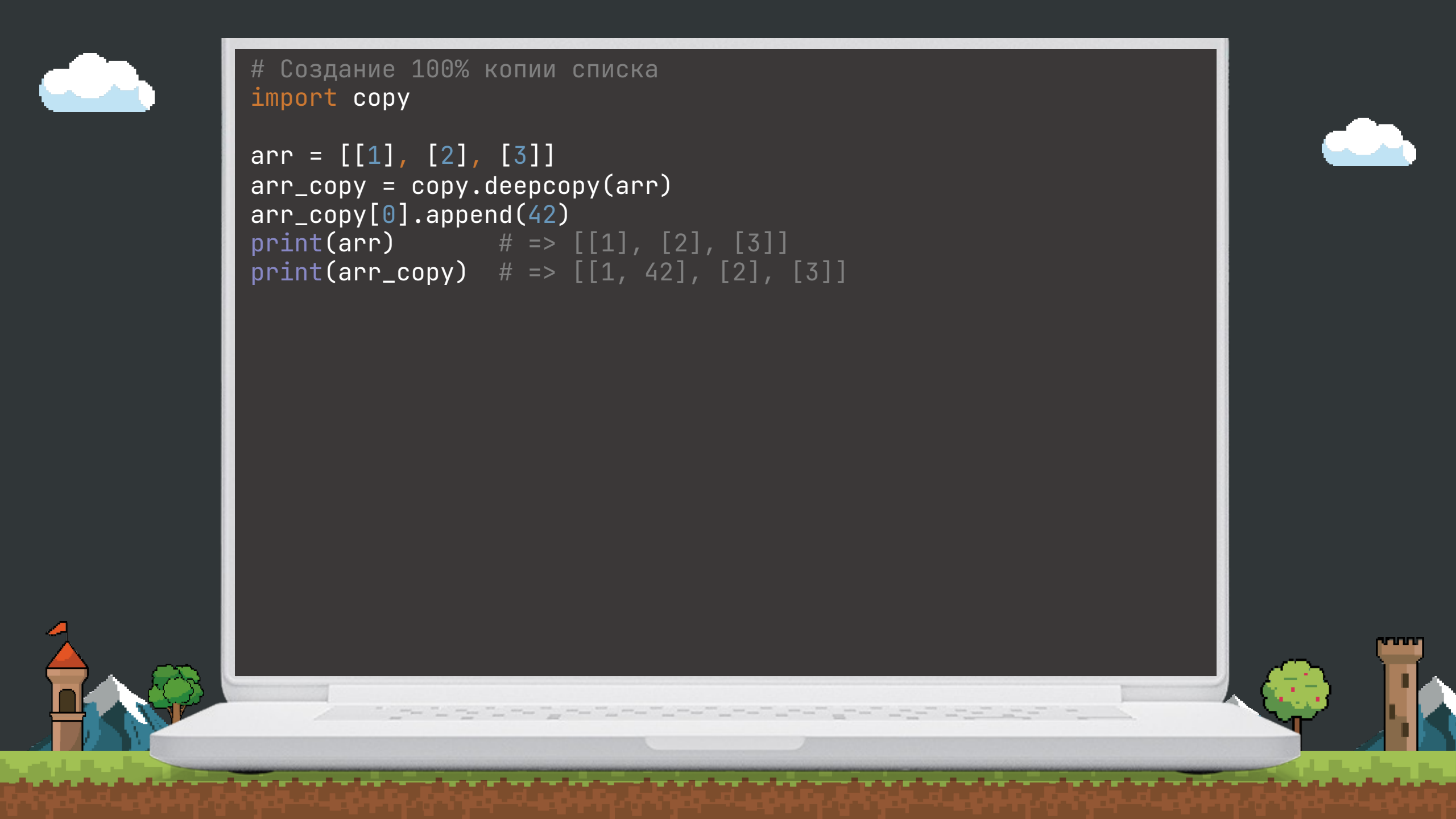
```
# Создание копии списка  
# Чтобы не изменить исходный список  
# иногда требуется сделать копию
```

```
arr = [1, 2, 3]  
arr_copy = arr[:]  
arr_copy[0] = 42  
print(arr)      # => [1, 2, 3]  
print(arr_copy) # => [42, 2, 3]
```



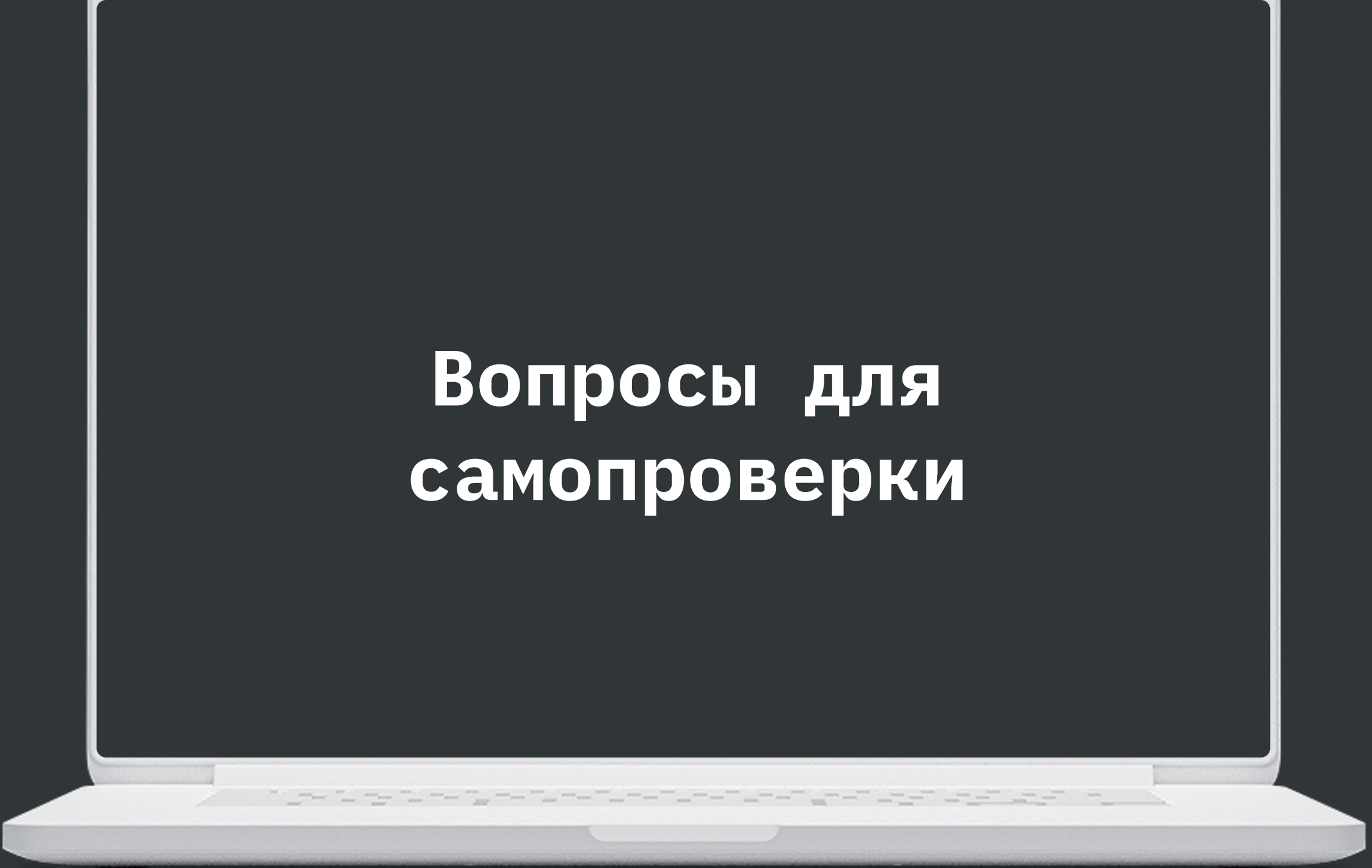
```
# Создание копии списка
# При создании копии списка, элементы нового списка – те же самые
# объекты, что элементы исходного списка (а не копии)
```

```
arr = [[1], [2], [3]]
arr_copy = arr[:]
arr_copy[0].append(42)
print(arr)      # => [[1, 42], [2], [3]]
print(arr_copy) # => [[1, 42], [2], [3]]
```

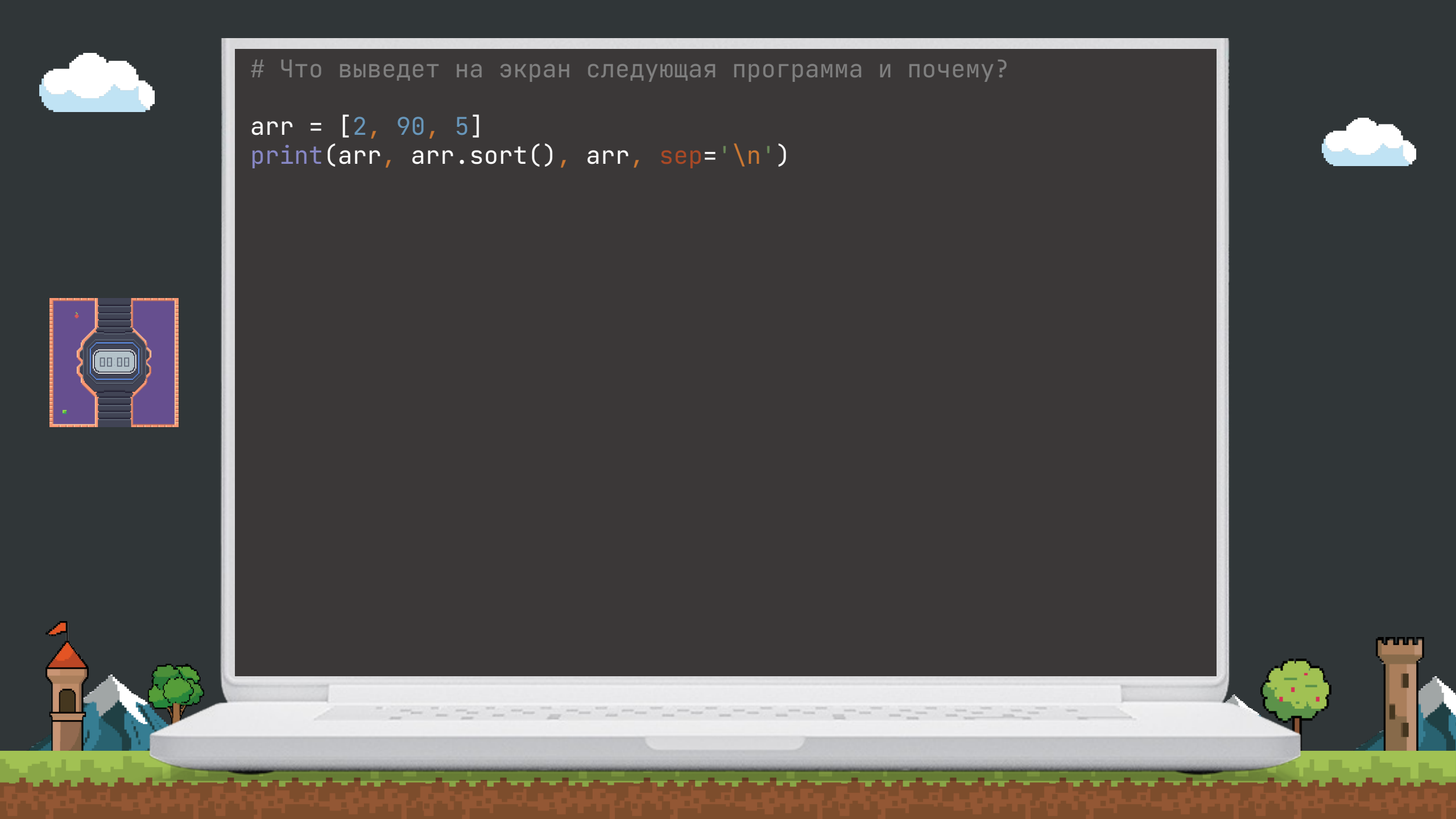


```
# Создание 100% копии списка
import copy

arr = [[1], [2], [3]]
arr_copy = copy.deepcopy(arr)
arr_copy[0].append(42)
print(arr)          # => [[1], [2], [3]]
print(arr_copy)     # => [[1, 42], [2], [3]]
```

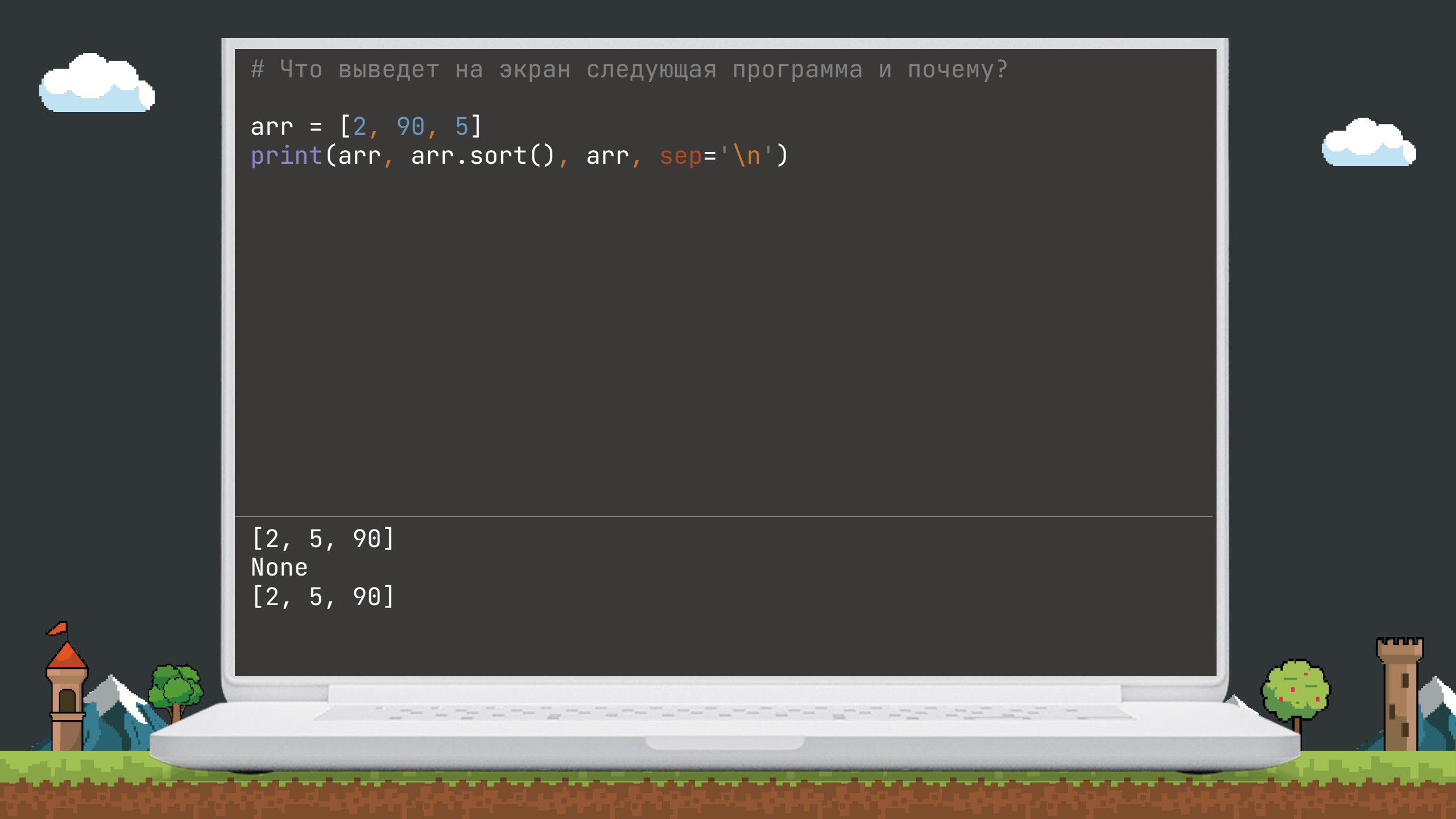
A white laptop is shown from a front-facing perspective, slightly angled. The screen is dark and displays the text "Вопросы для самопроверки" in white. The laptop's keyboard and trackpad are visible at the bottom.

Вопросы для самопроверки



Что выведет на экран следующая программа и почему?

```
arr = [2, 90, 5]  
print(arr, arr.sort(), arr, sep='\n')
```



```
# Что выведет на экран следующая программа и почему?
```

```
arr = [2, 90, 5]  
print(arr, arr.sort(), arr, sep='\n')
```

```
[2, 5, 90]
```

```
None
```

```
[2, 5, 90]
```




```
# Что выведет на экран следующая программа и почему?
```

```
x = 1
```

```
def double_x():  
    global x  
    x *= 2
```

```
print(x, double_x(), x, sep='\n')
```





```
# Что выведет на экран следующая программа и почему?
```

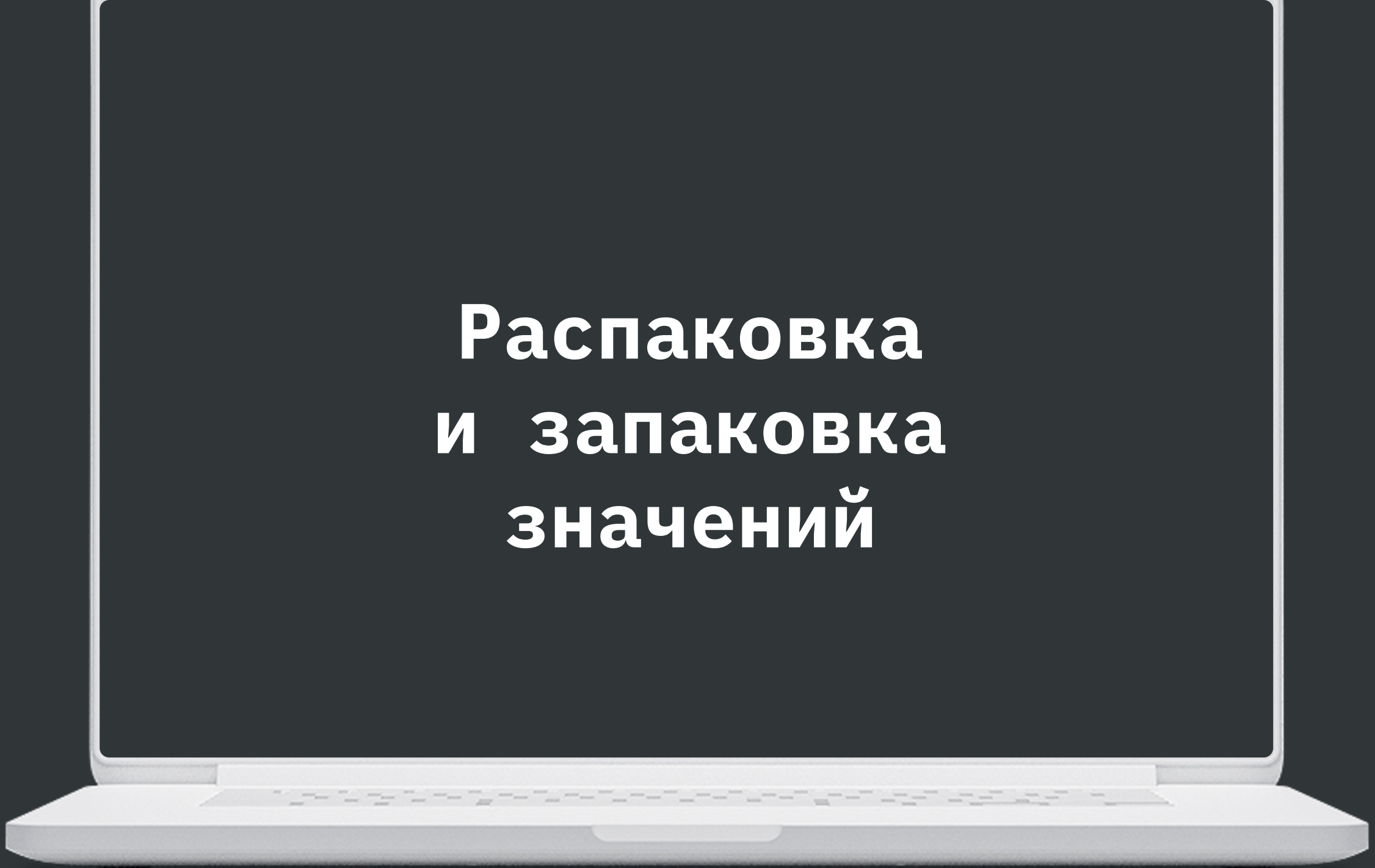
```
x = 1
```

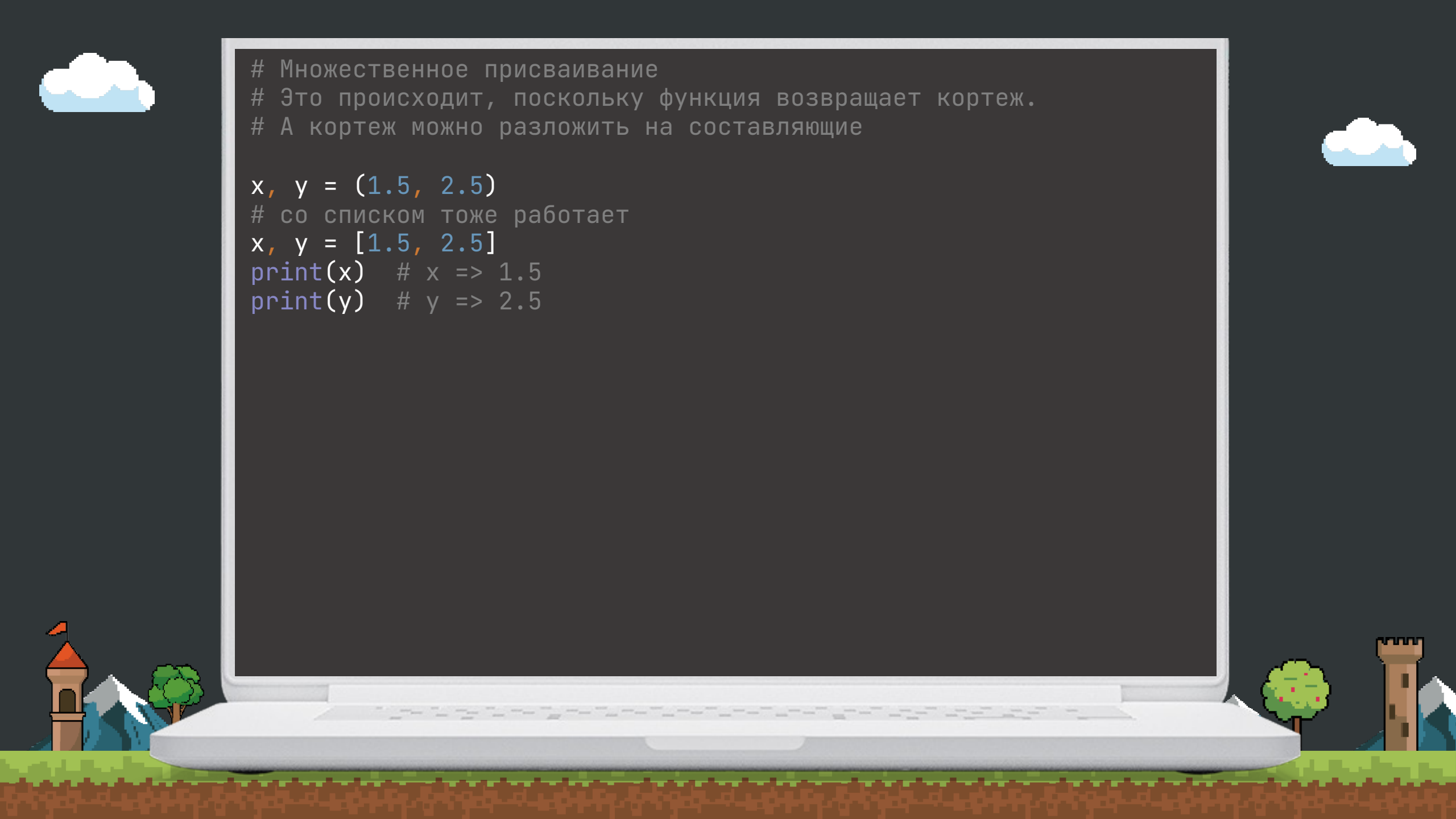
```
def double_x():  
    global x  
    x *= 2
```

```
print(x, double_x(), x, sep='\n')
```

```
1  
None  
2
```

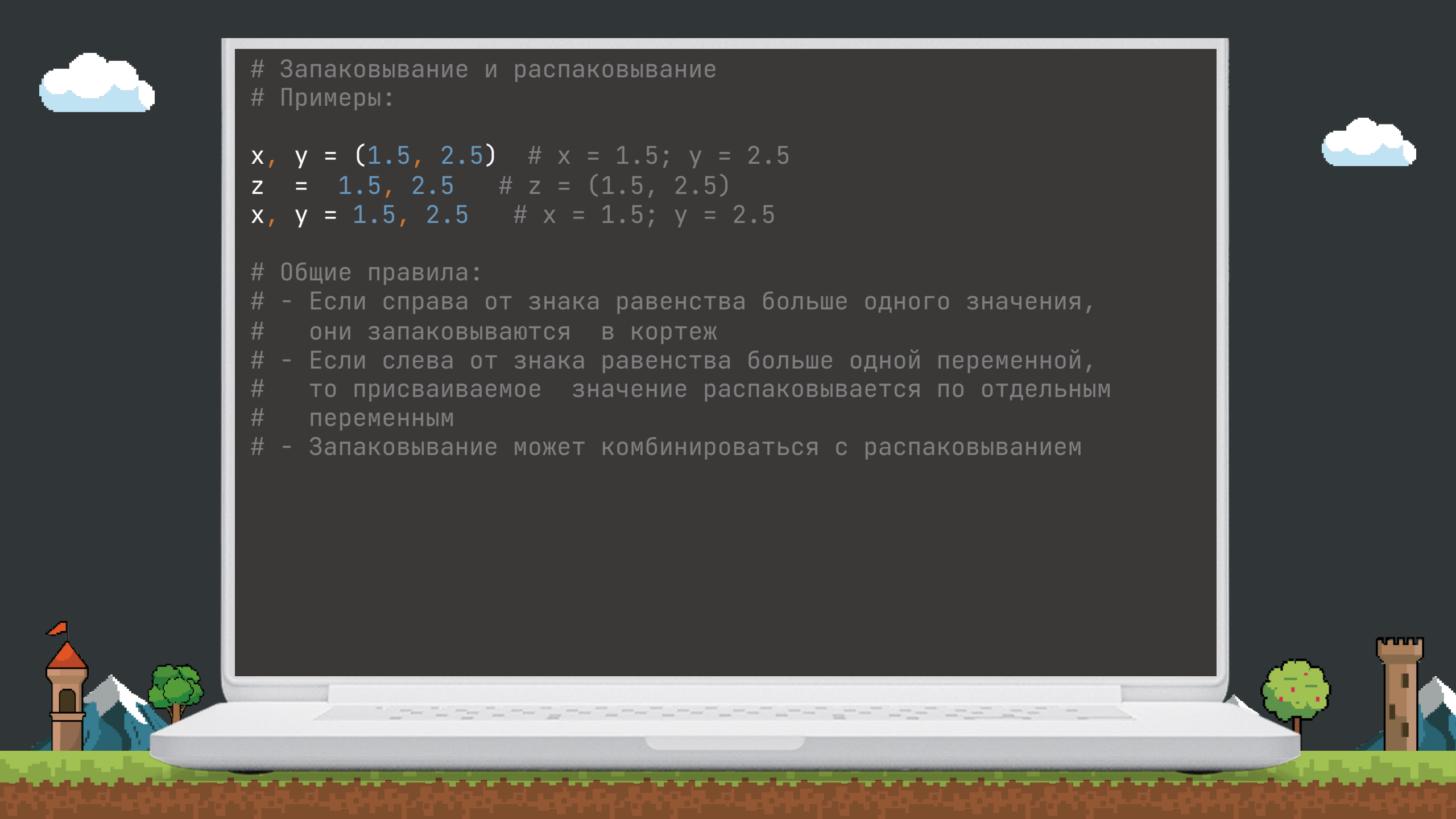
Распаковка и упаковка значений





```
# Множественное присваивание  
# Это происходит, поскольку функция возвращает кортеж.  
# А кортеж можно разложить на составляющие
```

```
x, y = (1.5, 2.5)  
# со списком тоже работает  
x, y = [1.5, 2.5]  
print(x)  # x => 1.5  
print(y)  # y => 2.5
```

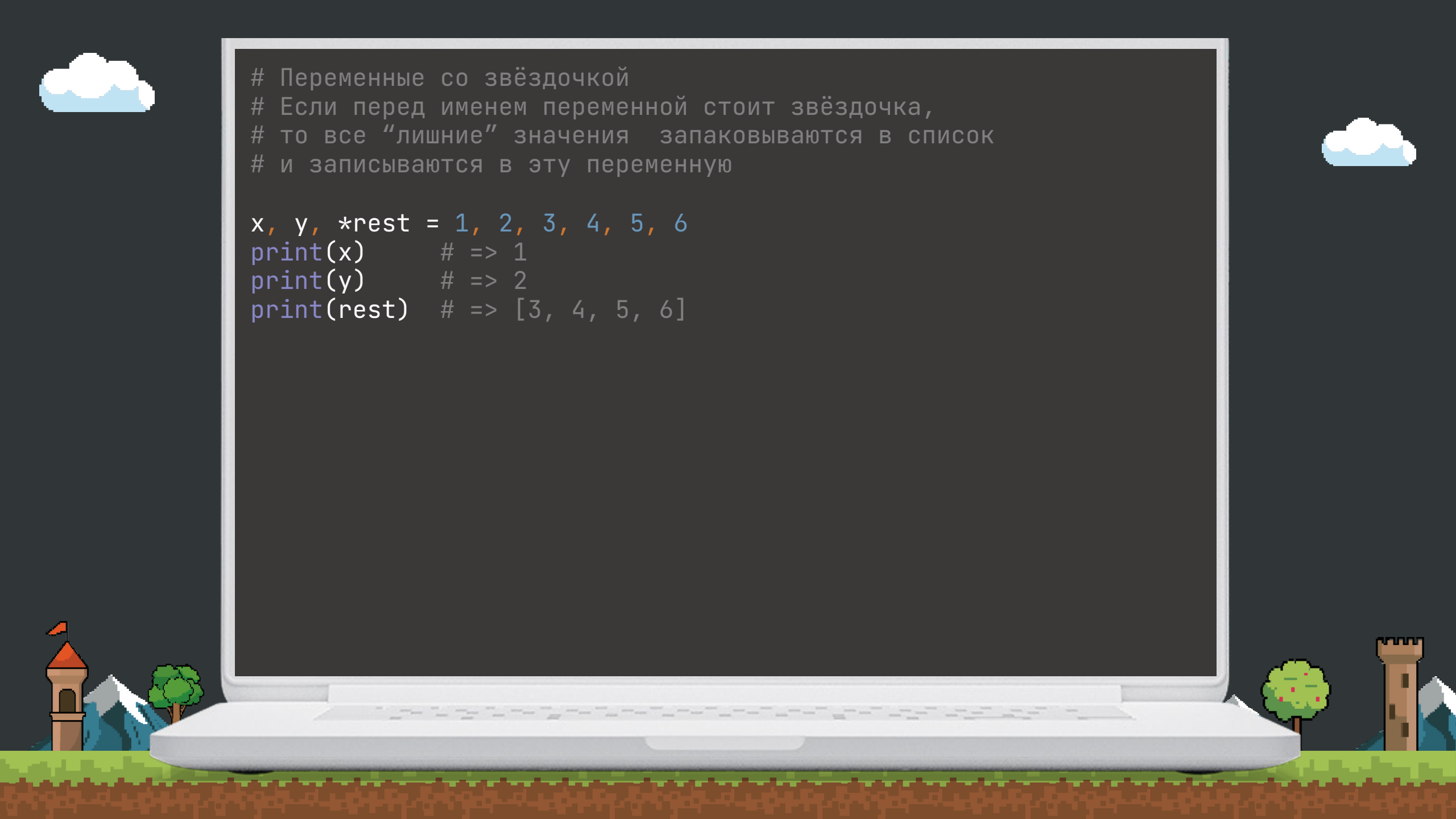


```
# Упаковывание и распаковывание  
# Примеры:
```

```
x, y = (1.5, 2.5) # x = 1.5; y = 2.5  
z = 1.5, 2.5     # z = (1.5, 2.5)  
x, y = 1.5, 2.5  # x = 1.5; y = 2.5
```

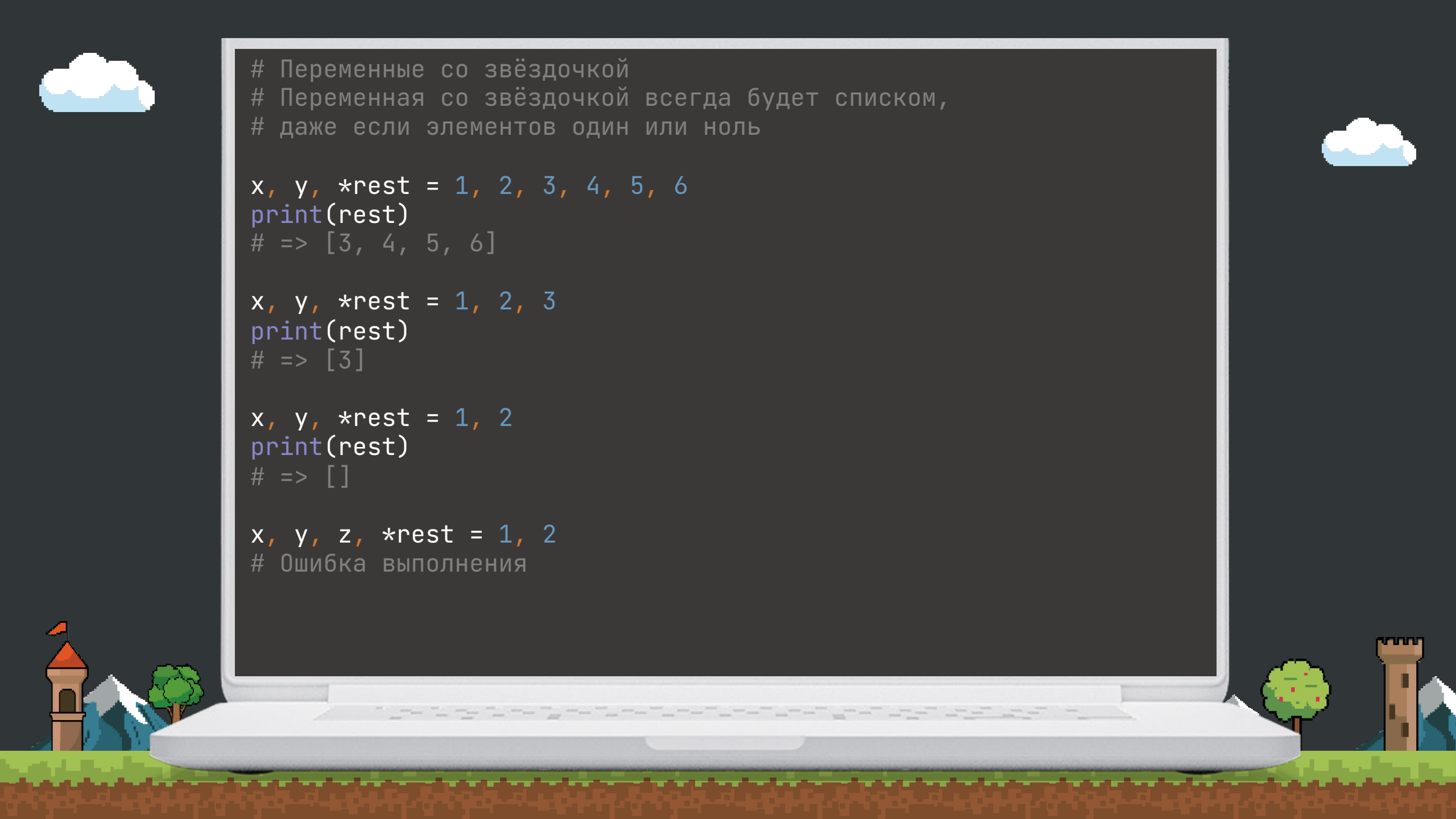
```
# Общие правила:
```

```
# - Если справа от знака равенства больше одного значения,  
#   они упаковываются в кортеж  
# - Если слева от знака равенства больше одной переменной,  
#   то присваиваемое значение распаковывается по отдельным  
#   переменным  
# - Упаковывание может комбинироваться с распаковыванием
```



```
# Переменные со звёздочкой  
# Если перед именем переменной стоит звёздочка,  
# то все "лишние" значения запаковываются в список  
# и записываются в эту переменную
```

```
x, y, *rest = 1, 2, 3, 4, 5, 6  
print(x)      # => 1  
print(y)      # => 2  
print(rest)   # => [3, 4, 5, 6]
```



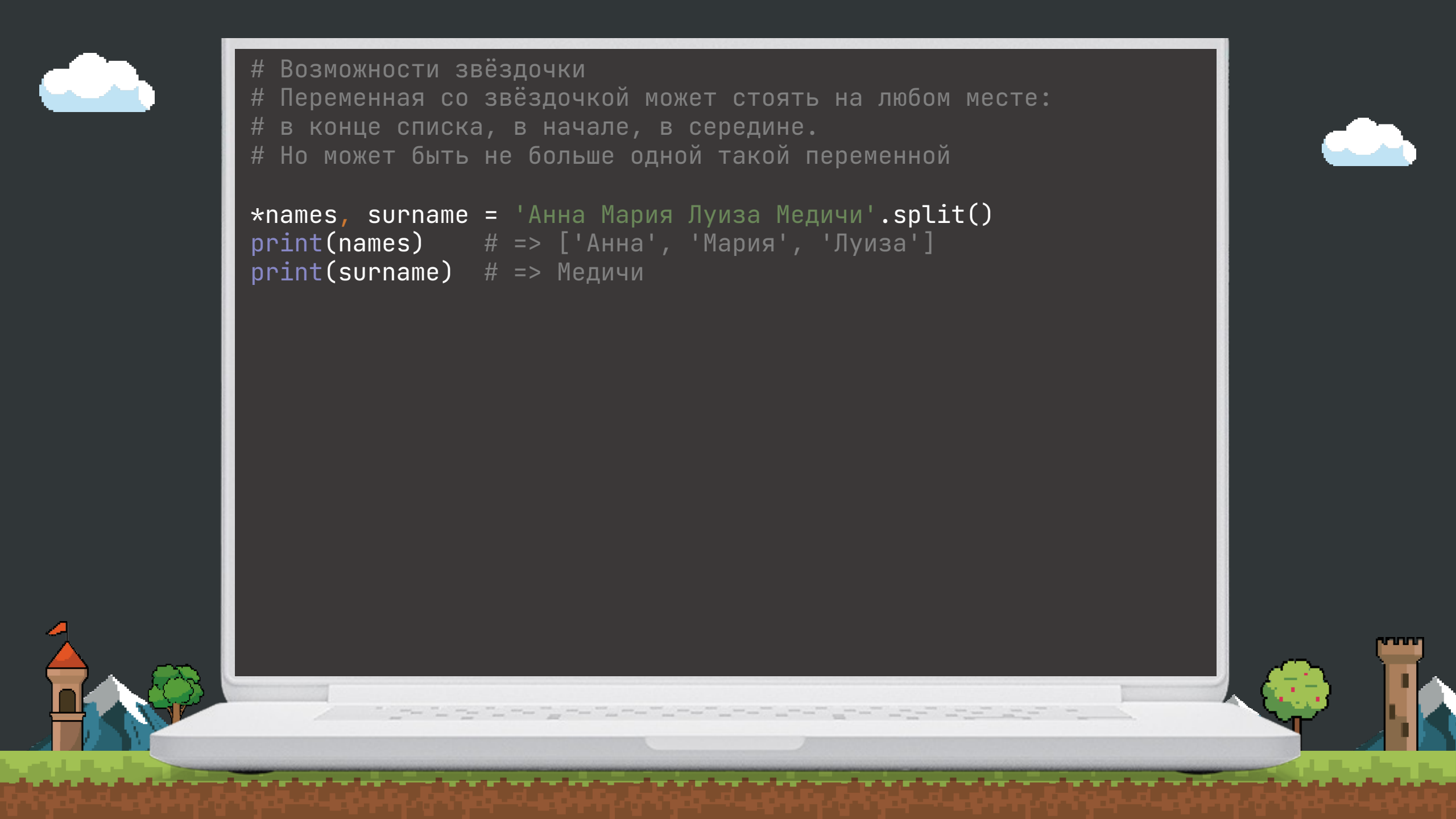
```
# Переменные со звёздочкой  
# Переменная со звёздочкой всегда будет списком,  
# даже если элементов один или ноль
```

```
x, y, *rest = 1, 2, 3, 4, 5, 6  
print(rest)  
# => [3, 4, 5, 6]
```

```
x, y, *rest = 1, 2, 3  
print(rest)  
# => [3]
```

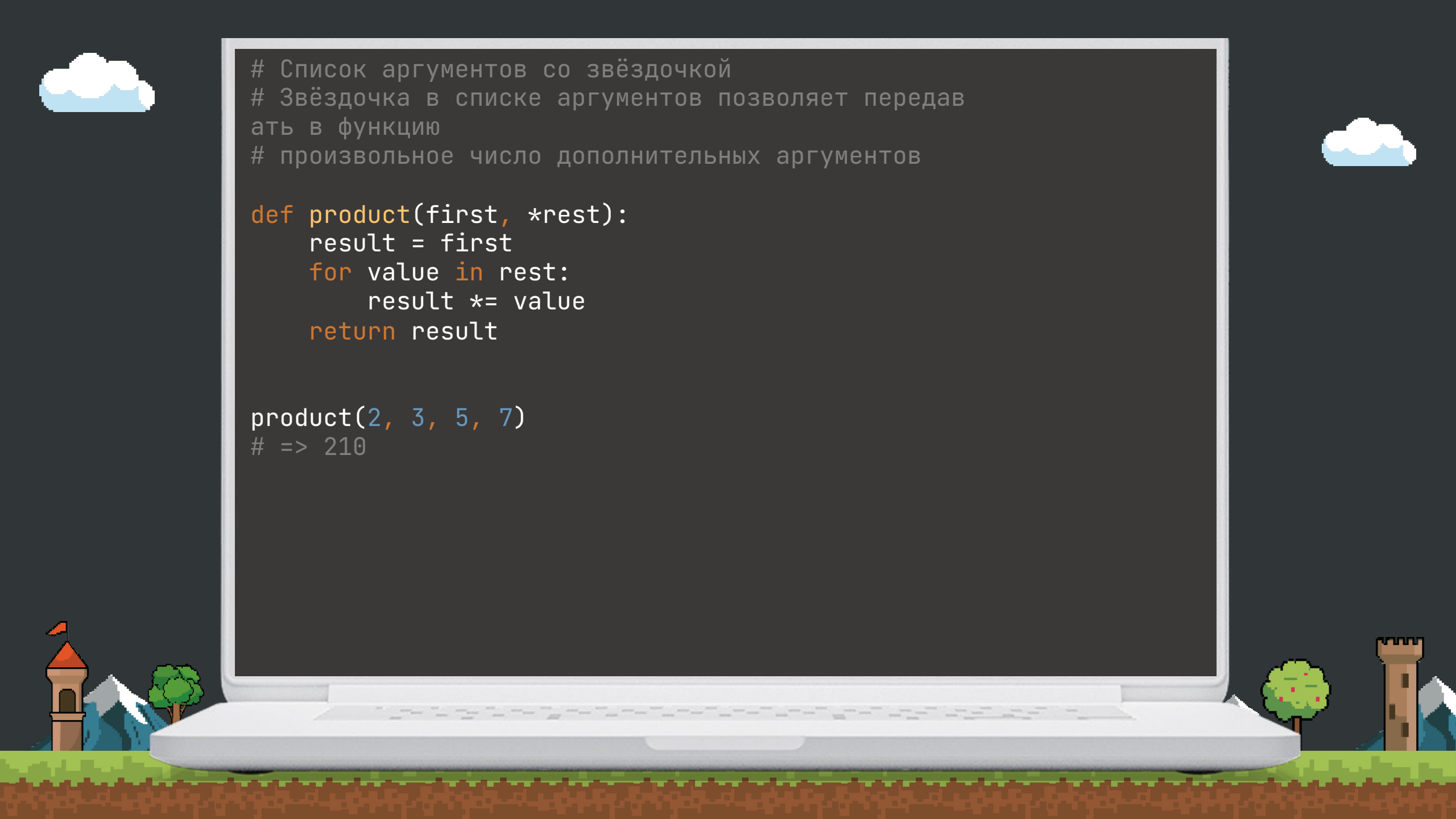
```
x, y, *rest = 1, 2  
print(rest)  
# => []
```

```
x, y, z, *rest = 1, 2  
# Ошибка выполнения
```



```
# Возможности звёздочки
# Переменная со звёздочкой может стоять на любом месте:
# в конце списка, в начале, в середине.
# Но может быть не больше одной такой переменной
```

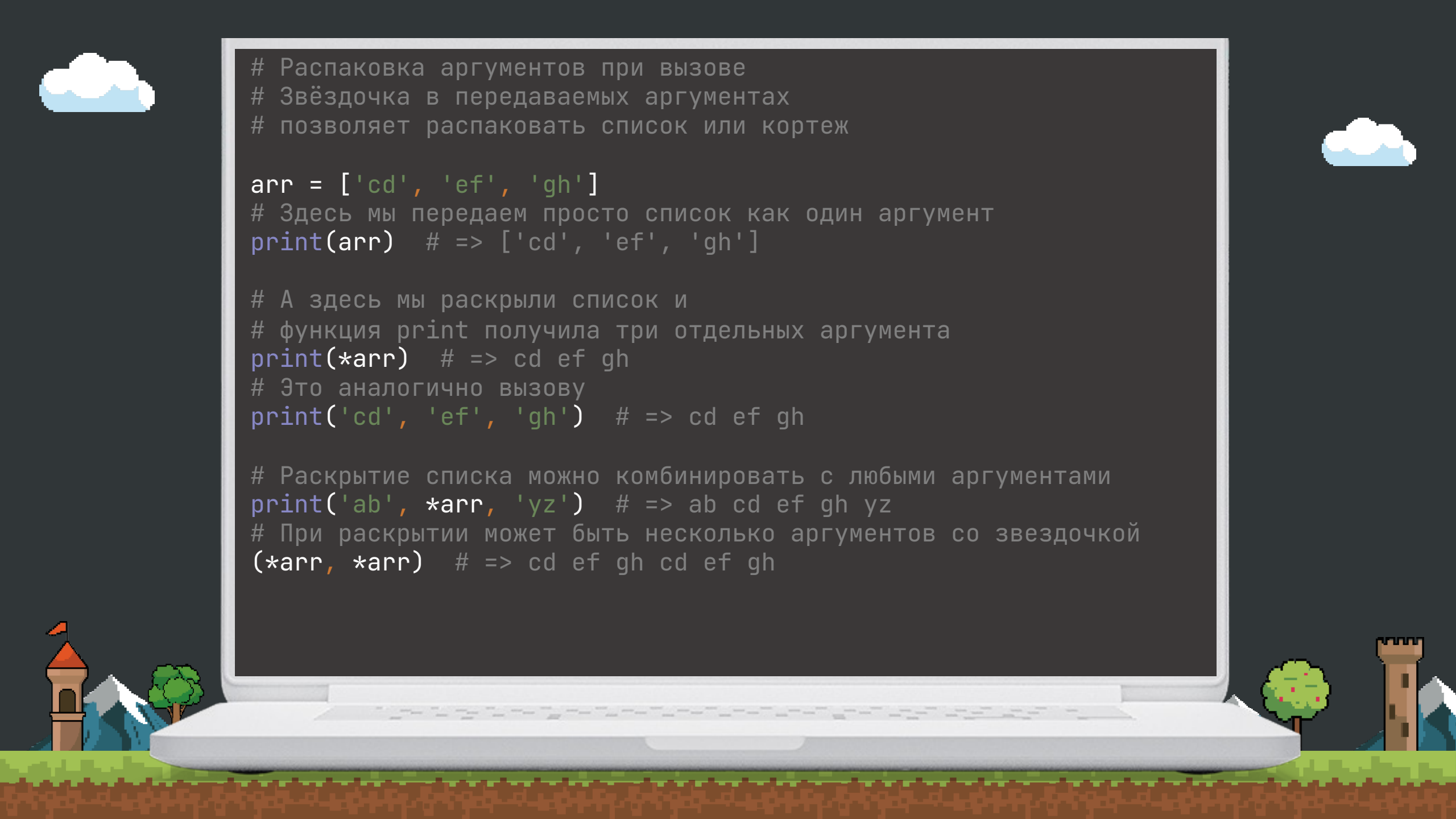
```
*names, surname = 'Анна Мария Луиза Медичи'.split()
print(names)      # => ['Анна', 'Мария', 'Луиза']
print(surname)    # => Медичи
```



```
# Список аргументов со звёздочкой
# Звёздочка в списке аргументов позволяет передав
ать в функцию
# произвольное число дополнительных аргументов
```

```
def product(first, *rest):
    result = first
    for value in rest:
        result *= value
    return result
```

```
product(2, 3, 5, 7)
# => 210
```

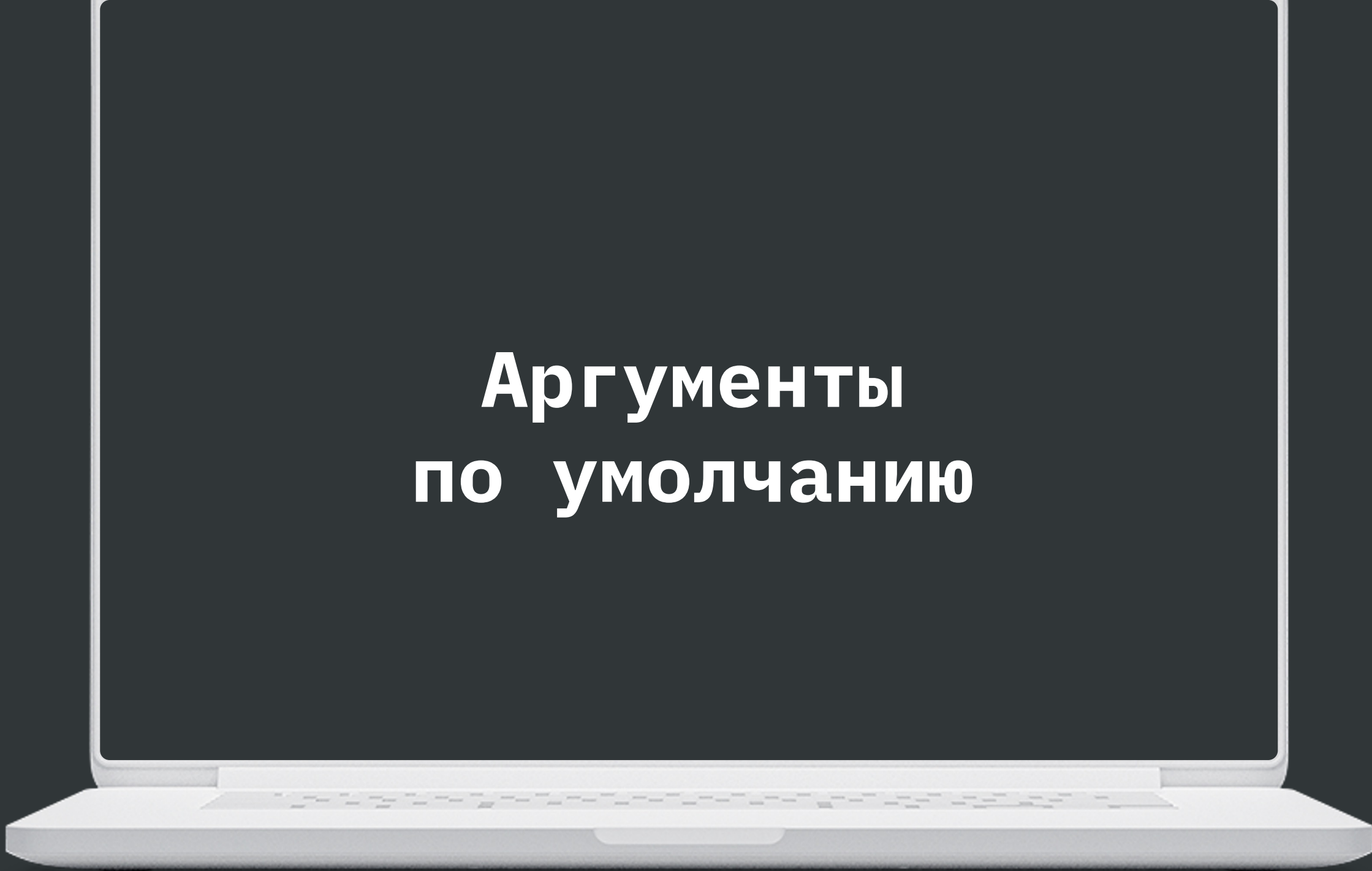
```
# Распаковка аргументов при вызове
# Звёздочка в передаваемых аргументах
# позволяет распаковать список или кортеж

arr = ['cd', 'ef', 'gh']
# Здесь мы передаем просто список как один аргумент
print(arr) # => ['cd', 'ef', 'gh']

# А здесь мы раскрыли список и
# функция print получила три отдельных аргумента
print(*arr) # => cd ef gh
# Это аналогично вызову
print('cd', 'ef', 'gh') # => cd ef gh

# Раскрытие списка можно комбинировать с любыми аргументами
print('ab', *arr, 'yz') # => ab cd ef gh yz
# При раскрытии может быть несколько аргументов со звездочкой
(*arr, *arr) # => cd ef gh cd ef gh
```

Аргументы по умолчанию





```
# Аргументы по умолчанию
```

```
int('101')      # => 101
```

```
int('101', 10)  # => 101
```

```
int('101', 2)   # => 5
```

```
# Функция `int` имеет два аргумента,  
# но второй из них по умолчанию равен 10  
# и его зачастую не указывают
```

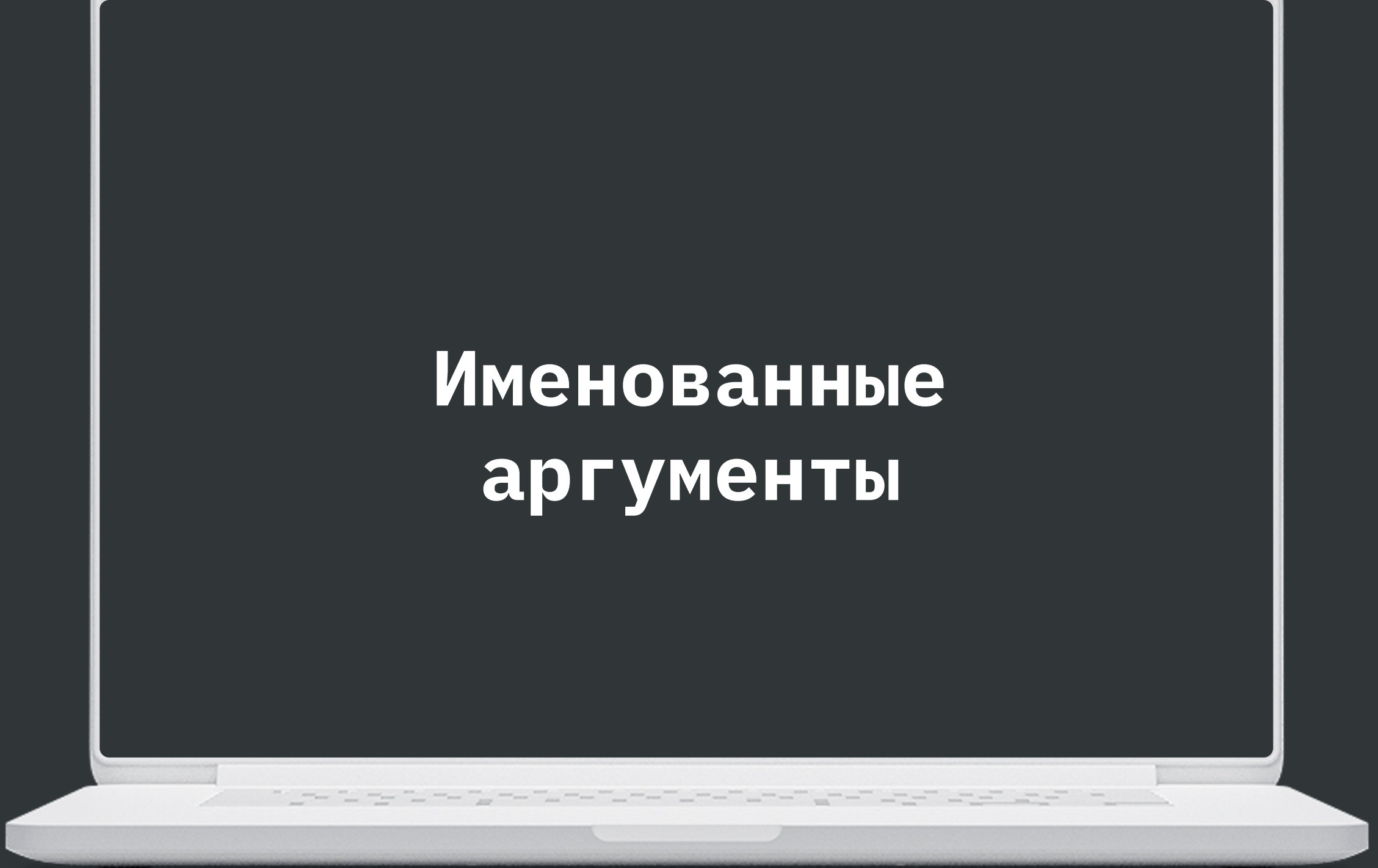


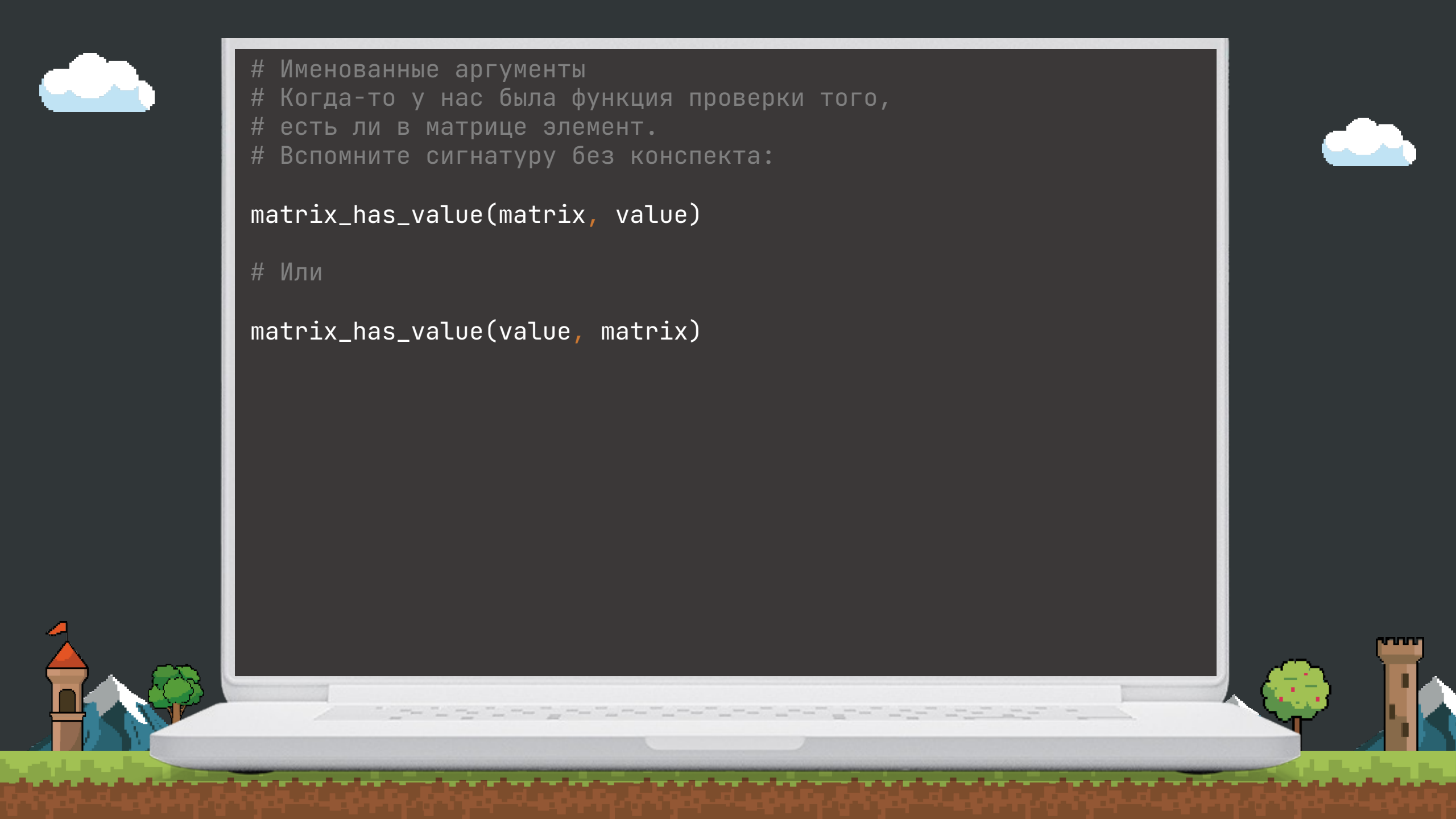
```
# Аргументы по умолчанию
```

```
def make_burger(typeOfMeat, withOnion=False, withTomato=True):  
    print('Булочка')  
    if withOnion:  
        print('Луковые колечки')  
    if withTomato:  
        print('Ломтик помидора')  
    print('Котлета из', typeOfMeat)  
    print('Булочка')
```

```
make_burger('курица') # бургер из курицы без лука, с помидорами  
make_burger('курица', True) # с луком и помидорами  
make_burger('курица', True, False) # с луком, без помидоров
```

Именованные аргументы



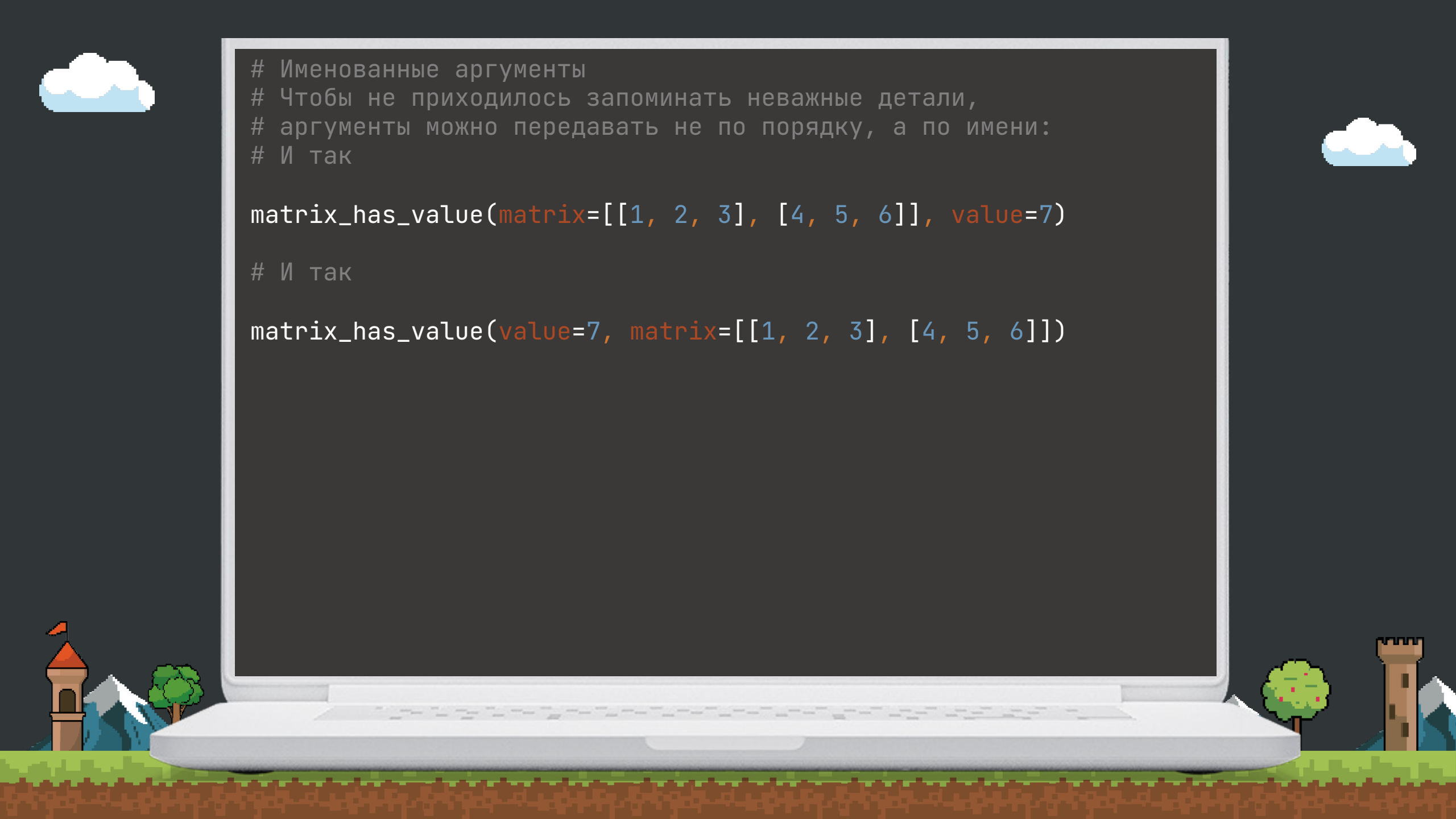


```
# Именованные аргументы  
# Когда-то у нас была функция проверки того,  
# есть ли в матрице элемент.  
# Вспомните сигнатуру без конспекта:
```

```
matrix_has_value(matrix, value)
```

```
# Или
```

```
matrix_has_value(value, matrix)
```

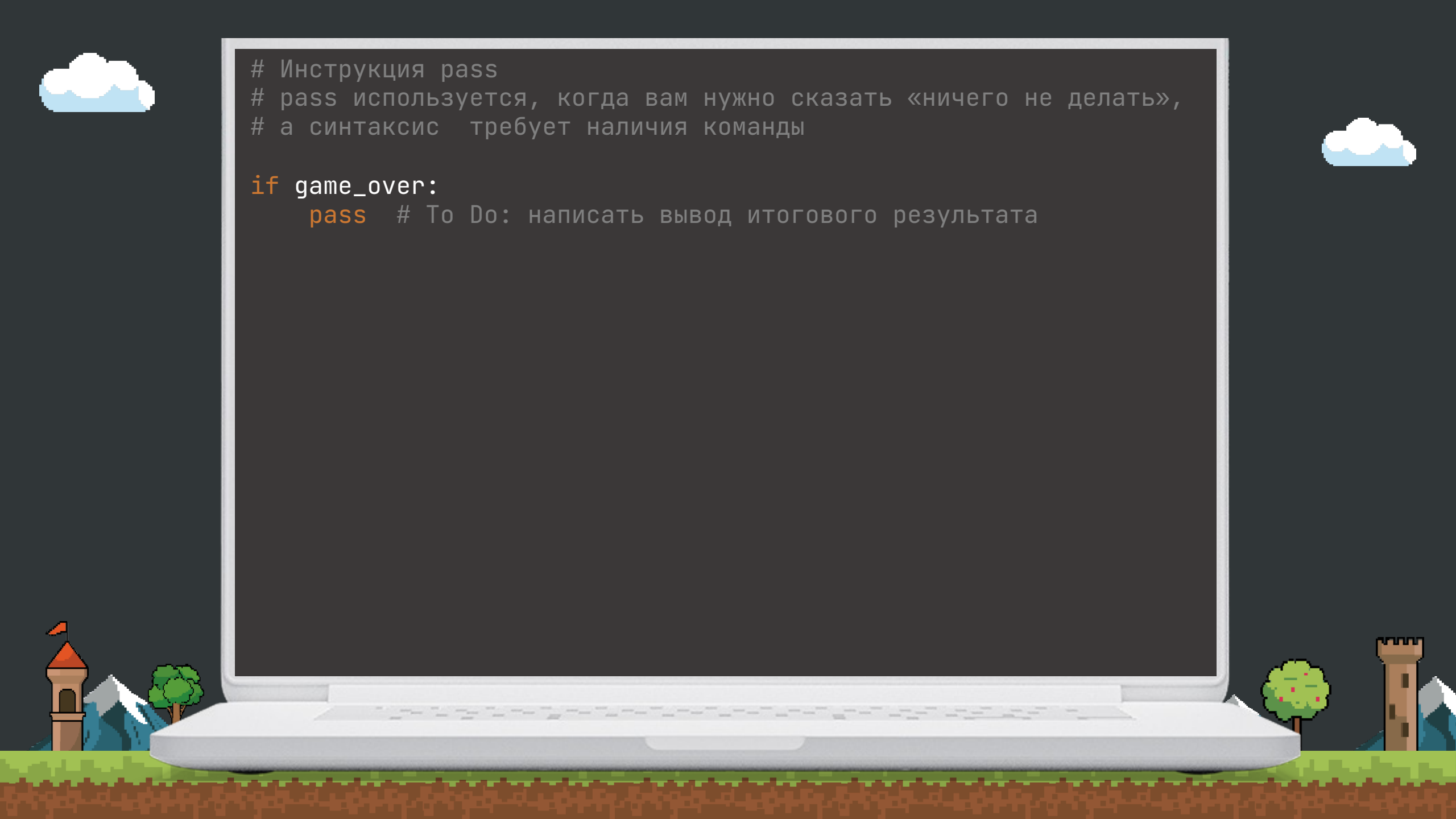


```
# Именованные аргументы  
# Чтобы не приходилось запоминать неважные детали,  
# аргументы можно передавать не по порядку, а по имени:  
# И так
```

```
matrix_has_value(matrix=[[1, 2, 3], [4, 5, 6]], value=7)
```

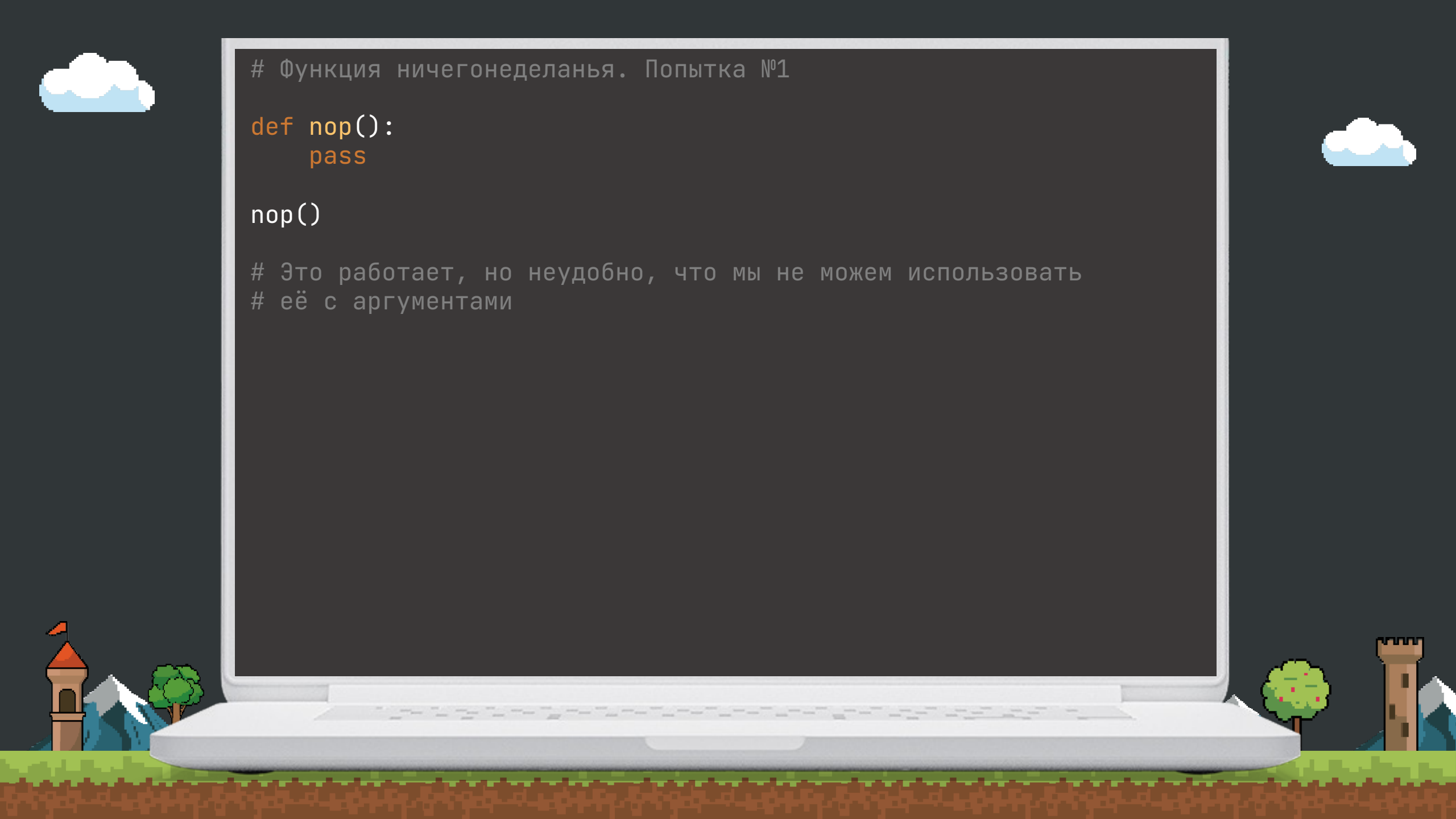
```
# И так
```

```
matrix_has_value(value=7, matrix=[[1, 2, 3], [4, 5, 6]])
```



```
# Инструкция pass  
# pass используется, когда вам нужно сказать «ничего не делать»,  
# а синтаксис требует наличия команды
```

```
if game_over:  
    pass # To Do: написать вывод итогового результата
```

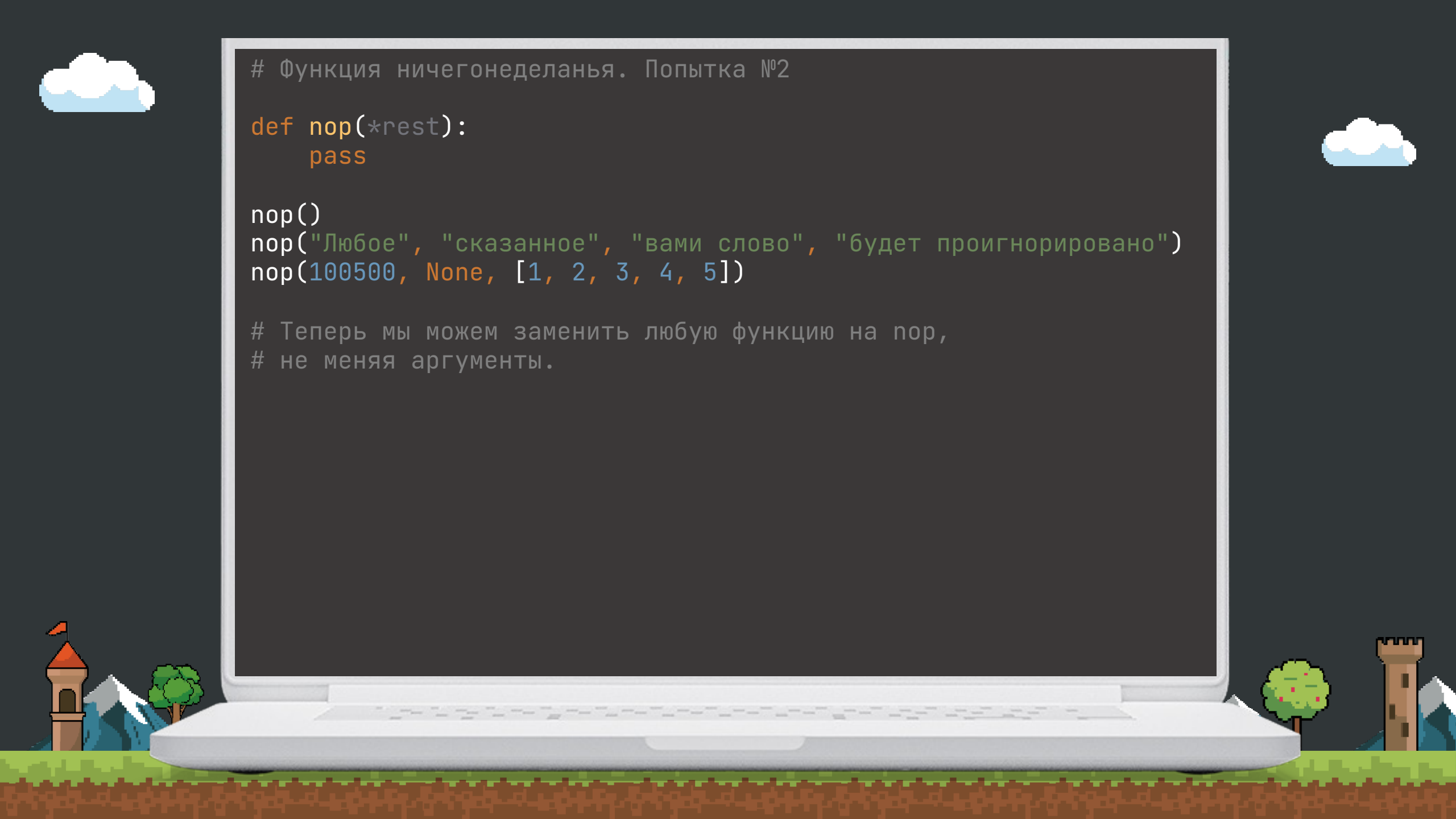



```
# Функция ничегонеделанья. Попытка №1
```

```
def nop():  
    pass
```

```
nop()
```

```
# Это работает, но неудобно, что мы не можем использовать  
# её с аргументами
```



```
# Функция ничегонеделанья. Попытка №2
```

```
def nop(*rest):  
    pass
```

```
nop()  
nop("Любое", "сказанное", "вами слово", "будет проигнорировано")  
nop(100500, None, [1, 2, 3, 4, 5])
```

```
# Теперь мы можем заменить любую функцию на nop,  
# не меняя аргументы.
```




```
# Задание: «сломайте» функцию nop  
# Придумайте такой набор аргументов,  
# чтобы print с ними работал, а nop – не работал
```

```
def nop(*rest):  
    pass
```

```
print(<???) # Работает  
nop(<???)  # Выдаёт ошибку
```



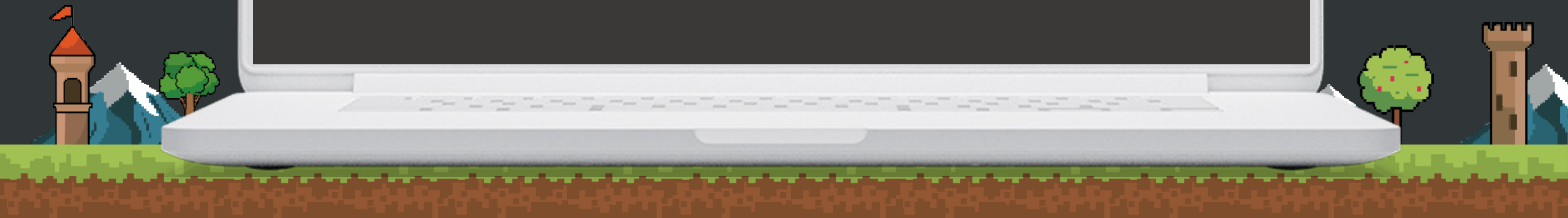


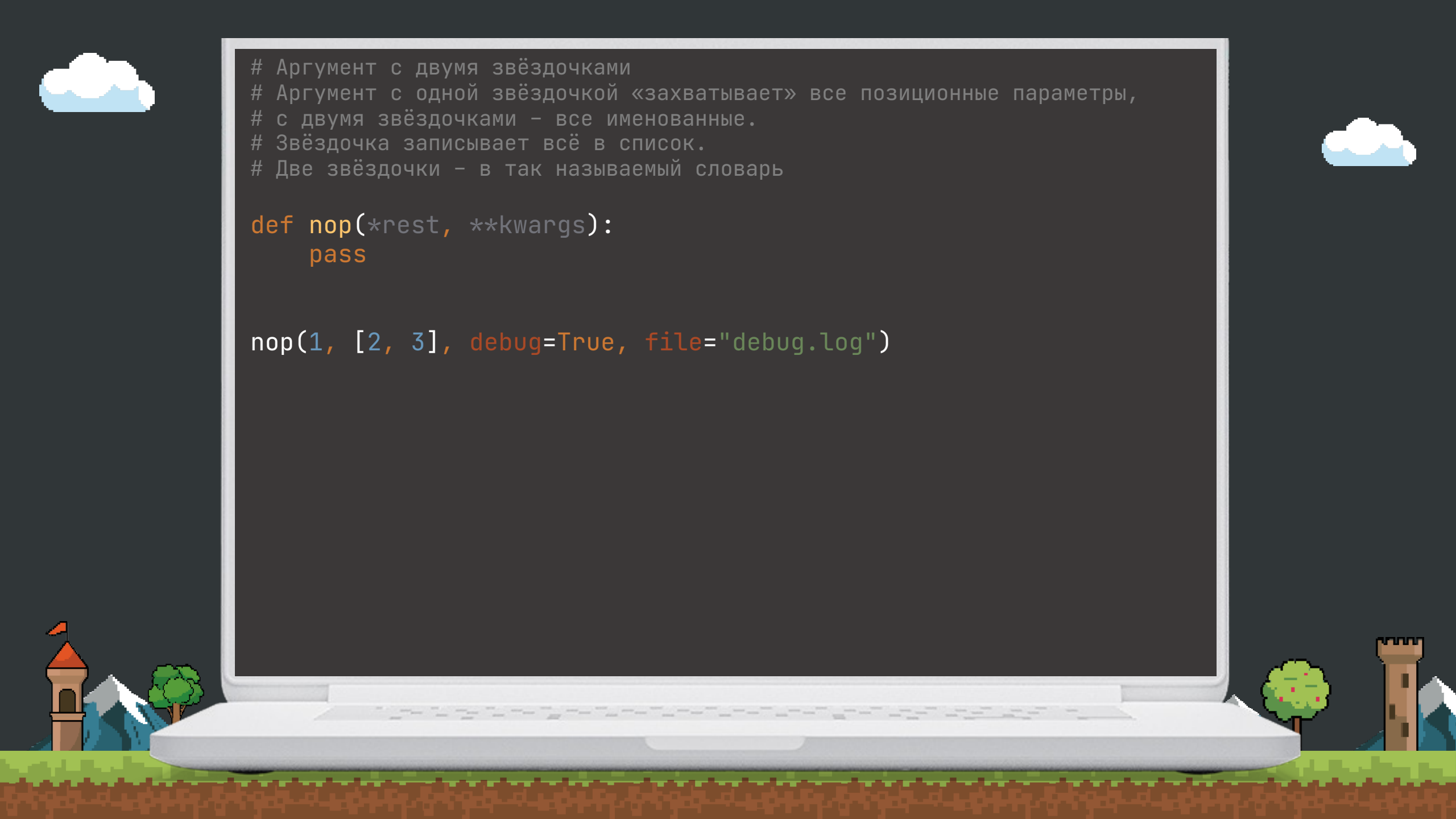
```
# Задание: «сломайте» функцию print
# Функция print не принимает именованные аргументы
```

```
print(1, 2, 3, sep=', ') # Работает
print(1, 2, 3, sep=', ') # Выдаёт ошибку
```

```
# Можно исправить так:
```

```
def print(*rest, sep=None, end=None):
    pass
```





```
# Аргумент с двумя звёздочками  
# Аргумент с одной звёздочкой «захватывает» все позиционные параметры,  
# с двумя звёздочками – все именованные.  
# Звёздочка записывает всё в список.  
# Две звёздочки – в так называемый словарь
```

```
def nop(*rest, **kwargs):  
    pass
```

```
nop(1, [2, 3], debug=True, file="debug.log")
```



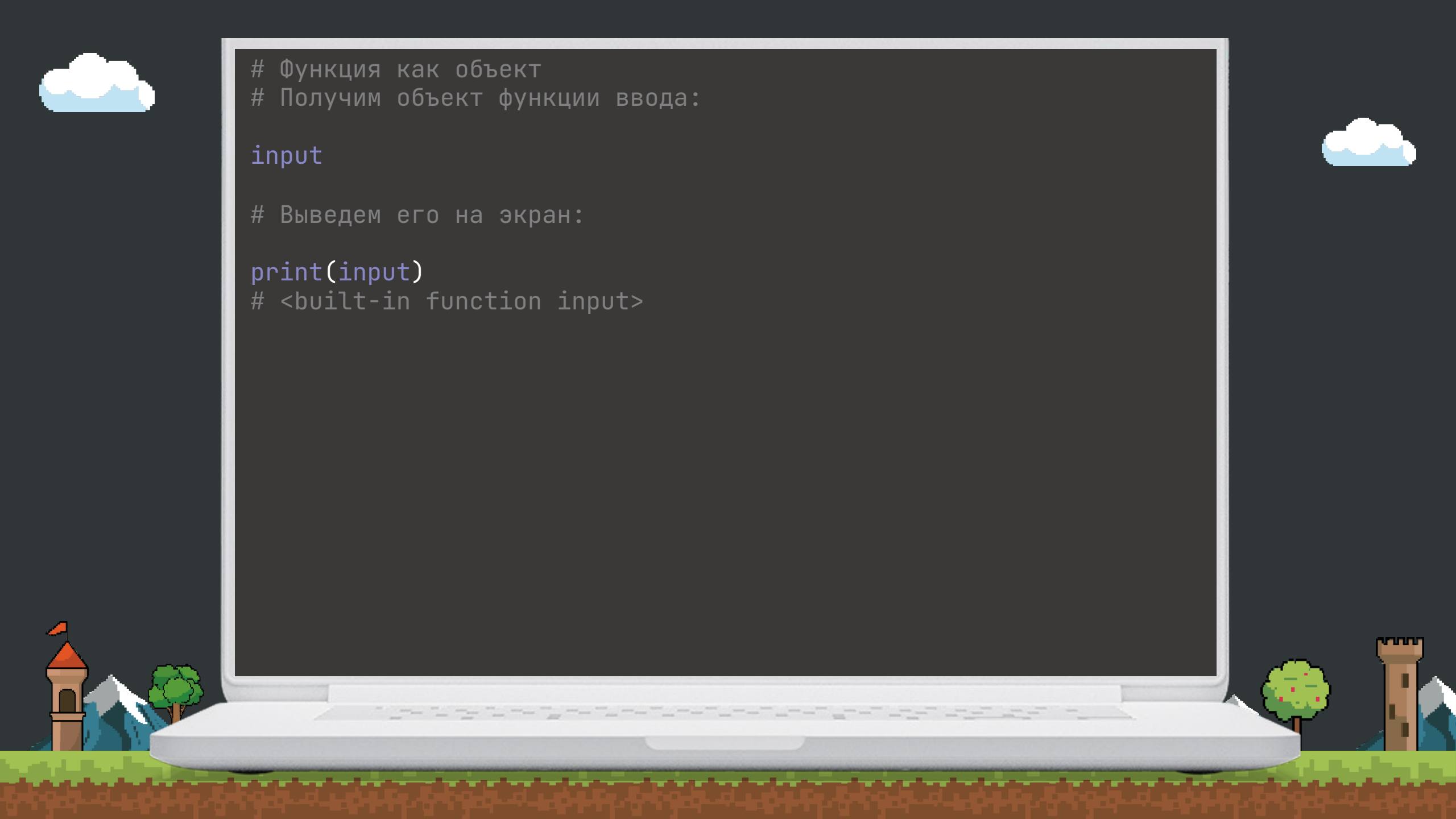
```
# Аргумент с двумя звёздочками
```

```
def profile(name, surname, city, *children, **additional_info):  
    print("Имя:", name)  
    print("Фамилия:", surname)  
    print("Город проживания:", city)  
    if len(children) > 0:  
        print("Дети:", ", ".join(children))  
    print(additional_info)  
  
profile("Сергей", "Михалков", "Москва", "Никита Михалков",  
        "Андрей Кончаловский", occupation="writer", diedIn=2009)
```

```
Имя: Сергей  
Фамилия: Михалков  
Город проживания: Москва  
Дети: Никита Михалков, Андрей Кончаловский  
{'occupation': 'writer', 'diedIn': 2009}
```

A white laptop is shown from a front-facing perspective, slightly angled. The screen is dark and displays the text 'Функция как объект' in a bold, white, sans-serif font. The laptop's keyboard and trackpad are visible at the bottom.

Функция как объект

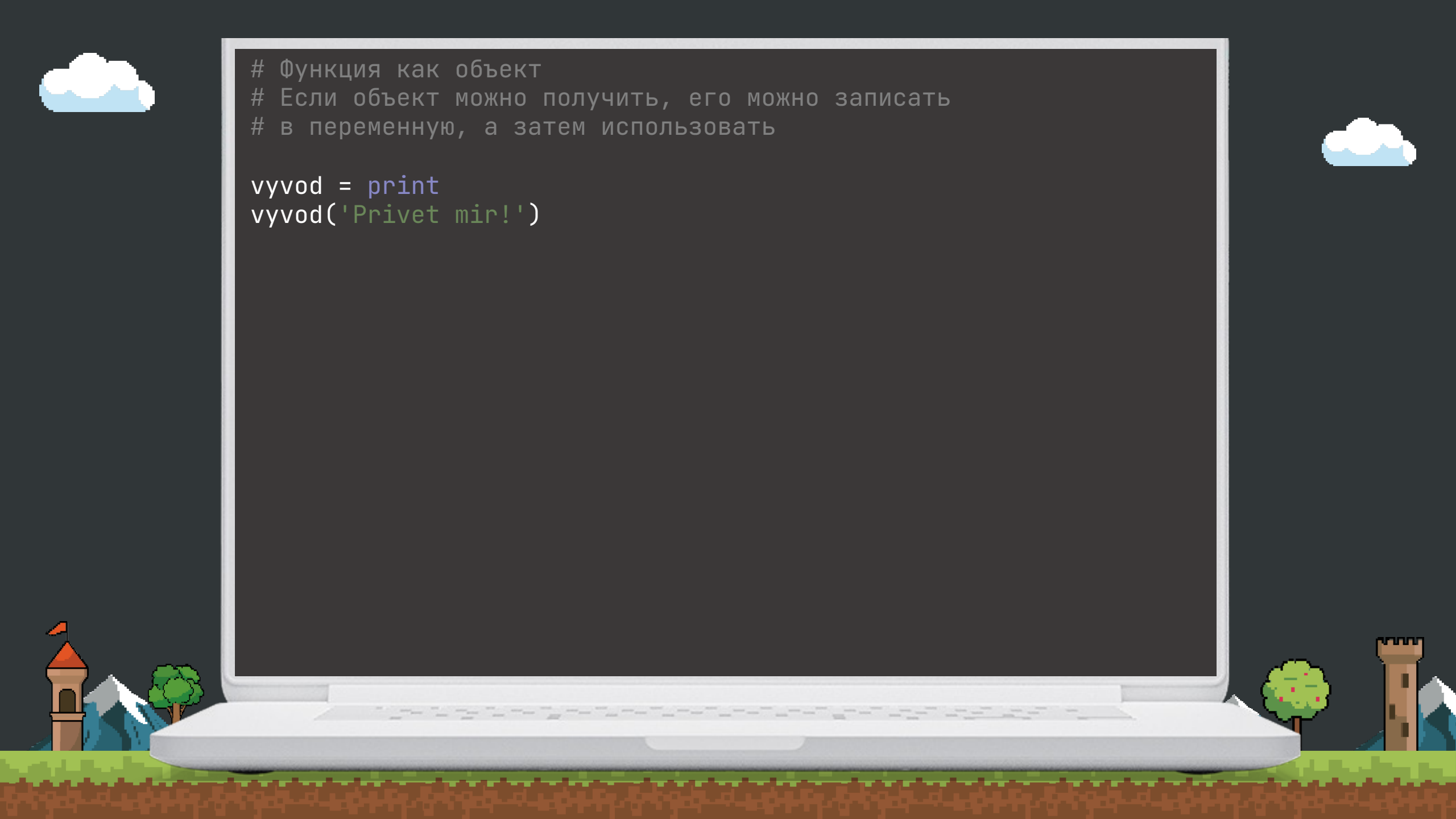


```
# Функция как объект  
# Получим объект функции ввода:
```

```
input
```

```
# Выведем его на экран:
```

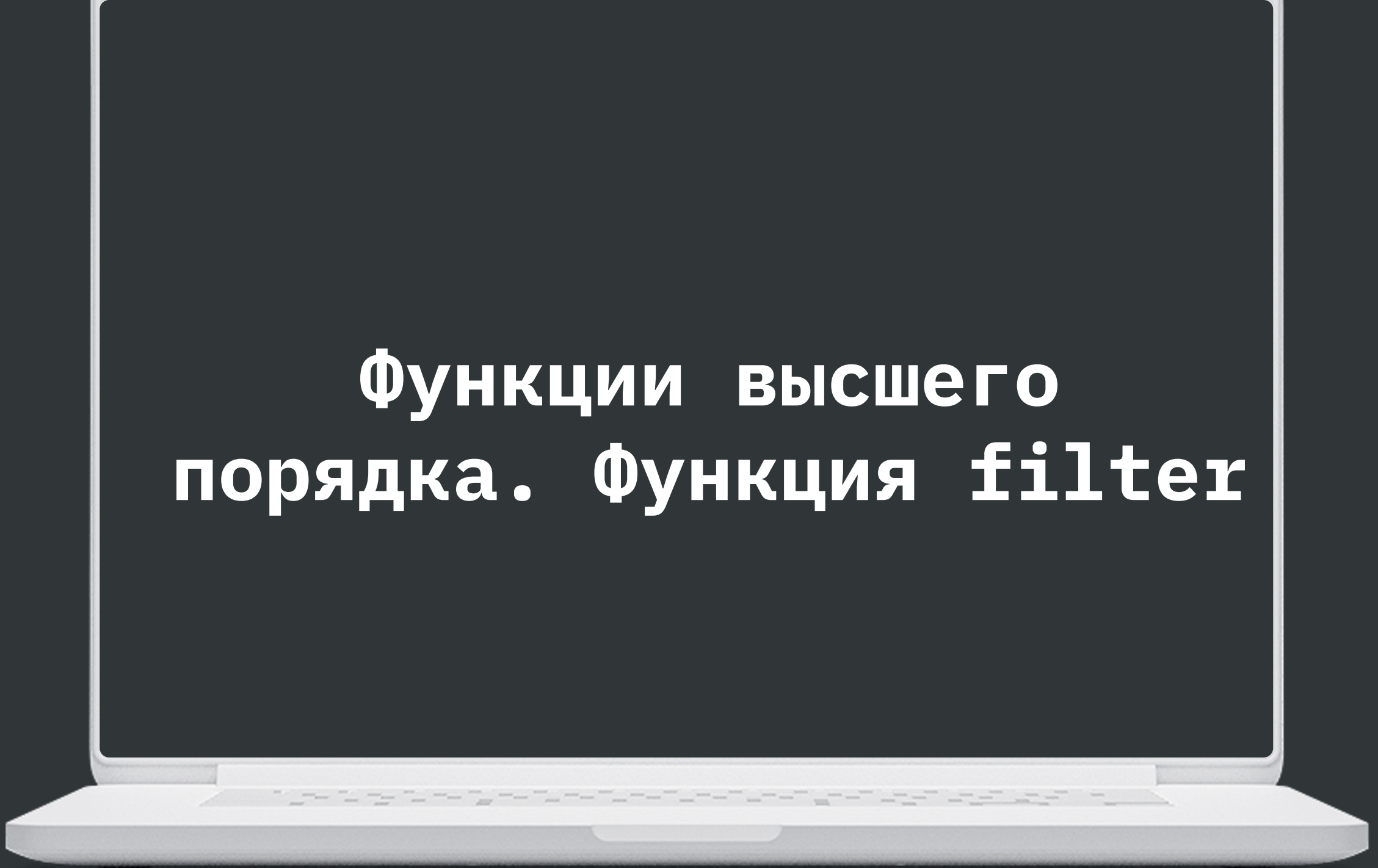
```
print(input)  
# <built-in function input>
```

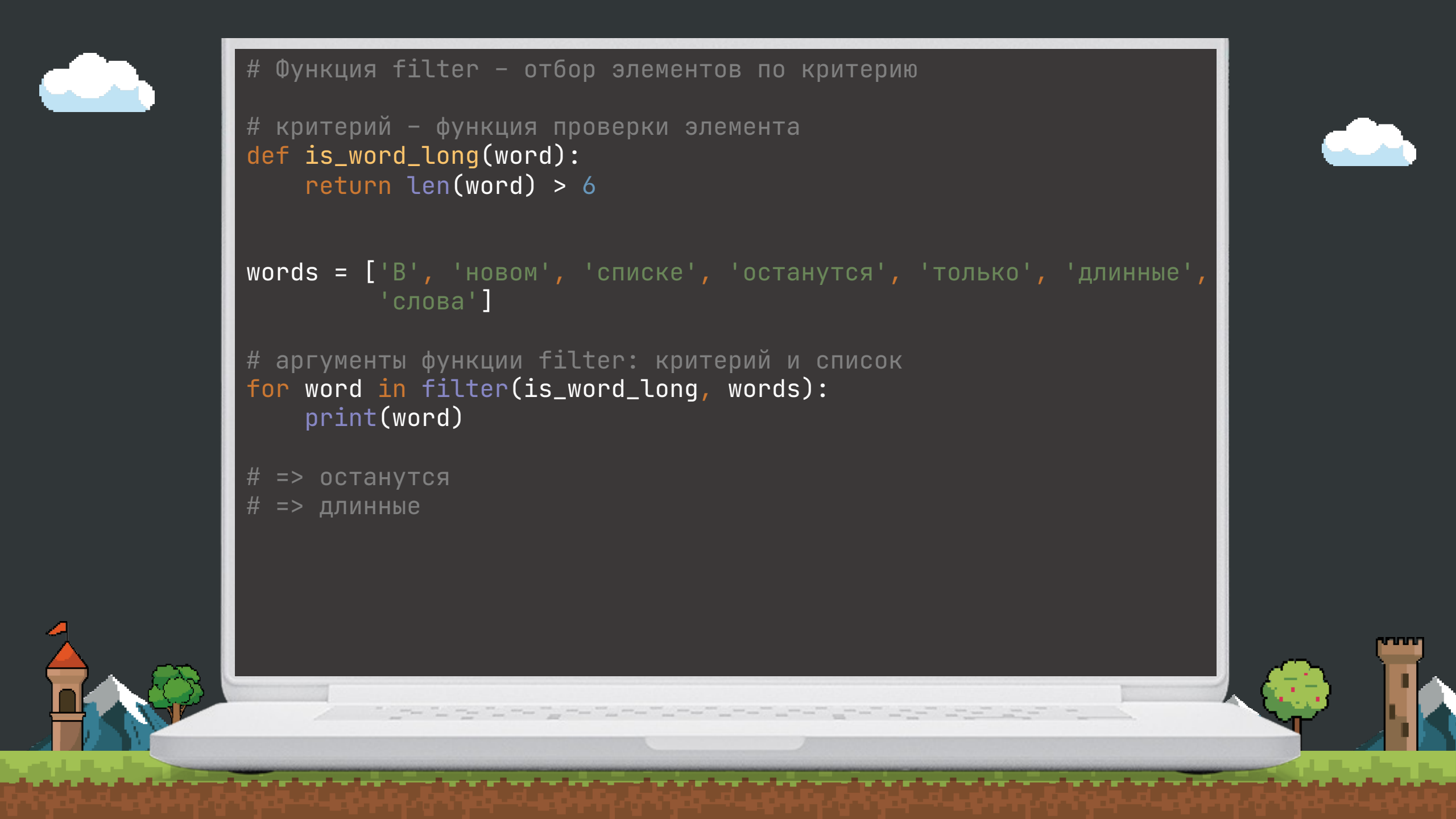



```
# Функция как объект  
# Если объект можно получить, его можно записать  
# в переменную, а затем использовать
```

```
vyvod = print  
vyvod('Privet mir!')
```

**Функции высшего
порядка. Функция `filter`**





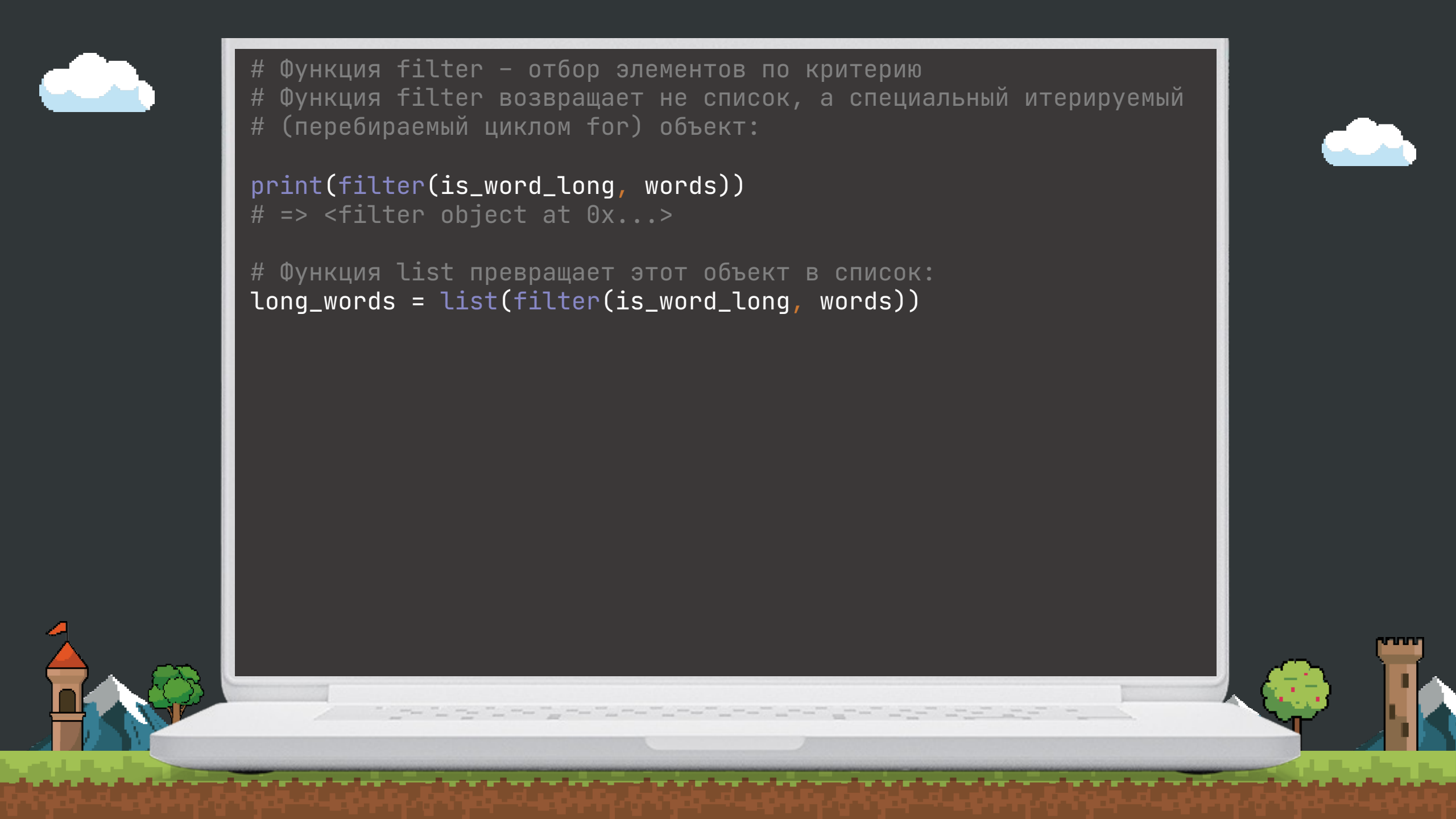
```
# Функция filter – отбор элементов по критерию

# критерий – функция проверки элемента
def is_word_long(word):
    return len(word) > 6

words = ['В', 'новом', 'списке', 'останутся', 'только', 'длинные',
        'слова']

# аргументы функции filter: критерий и список
for word in filter(is_word_long, words):
    print(word)

# => останутся
# => длинные
```

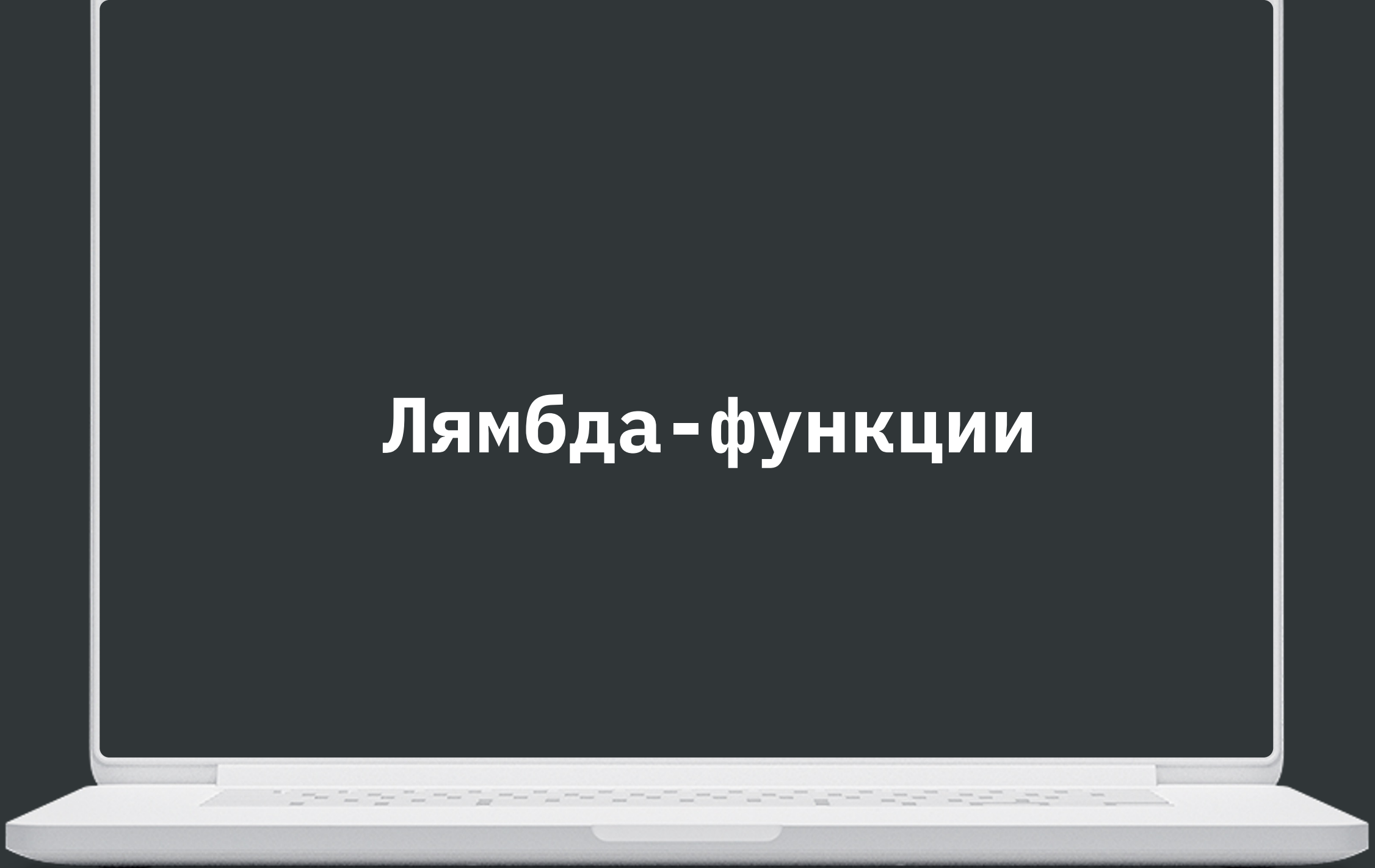


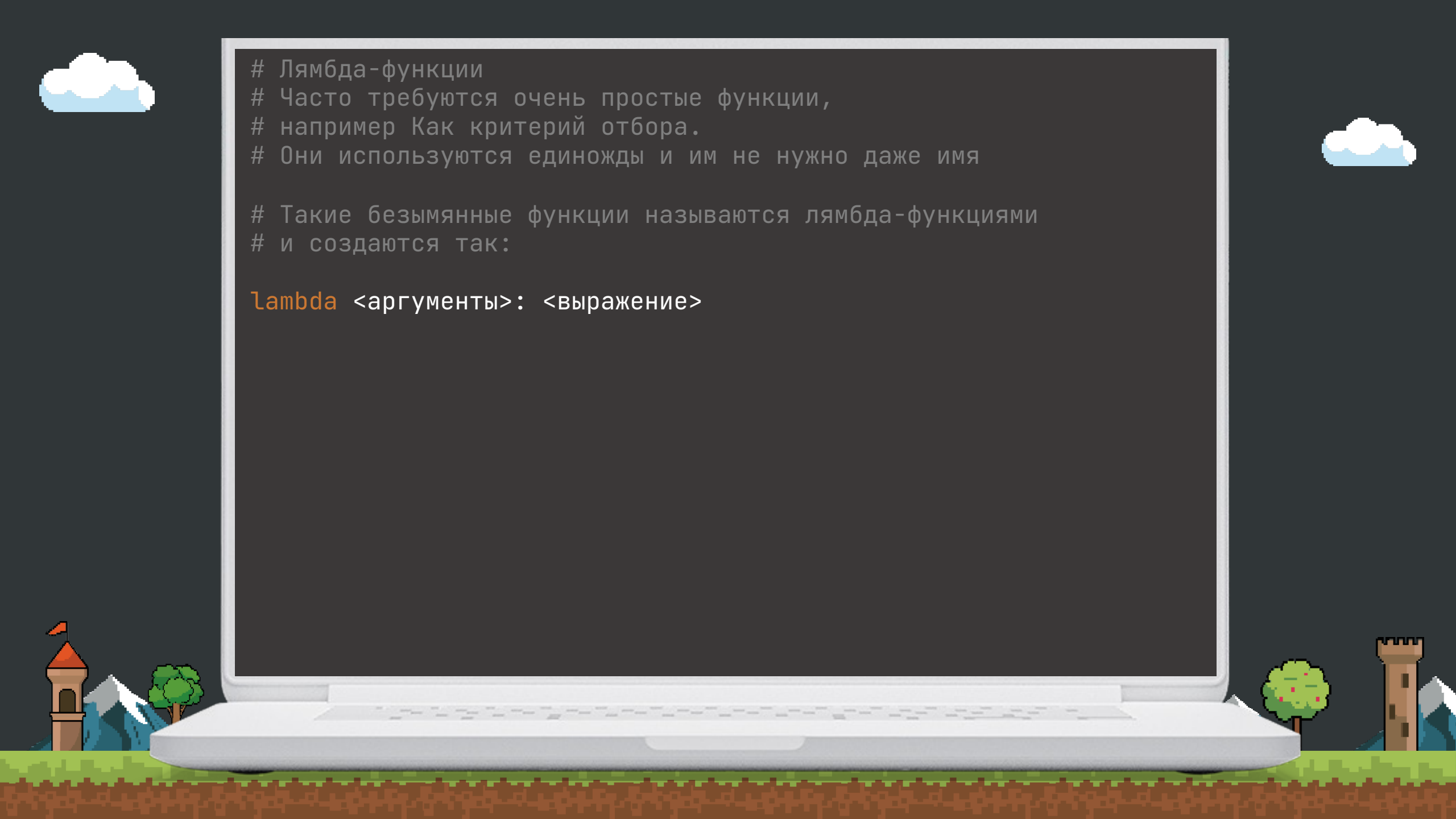
```
# Функция filter – отбор элементов по критерию  
# Функция filter возвращает не список, а специальный итерируемый  
# (перебираемый циклом for) объект:
```

```
print(filter(is_word_long, words))  
# => <filter object at 0x...>
```

```
# Функция list превращает этот объект в список:  
long_words = list(filter(is_word_long, words))
```

Лямбда-функции

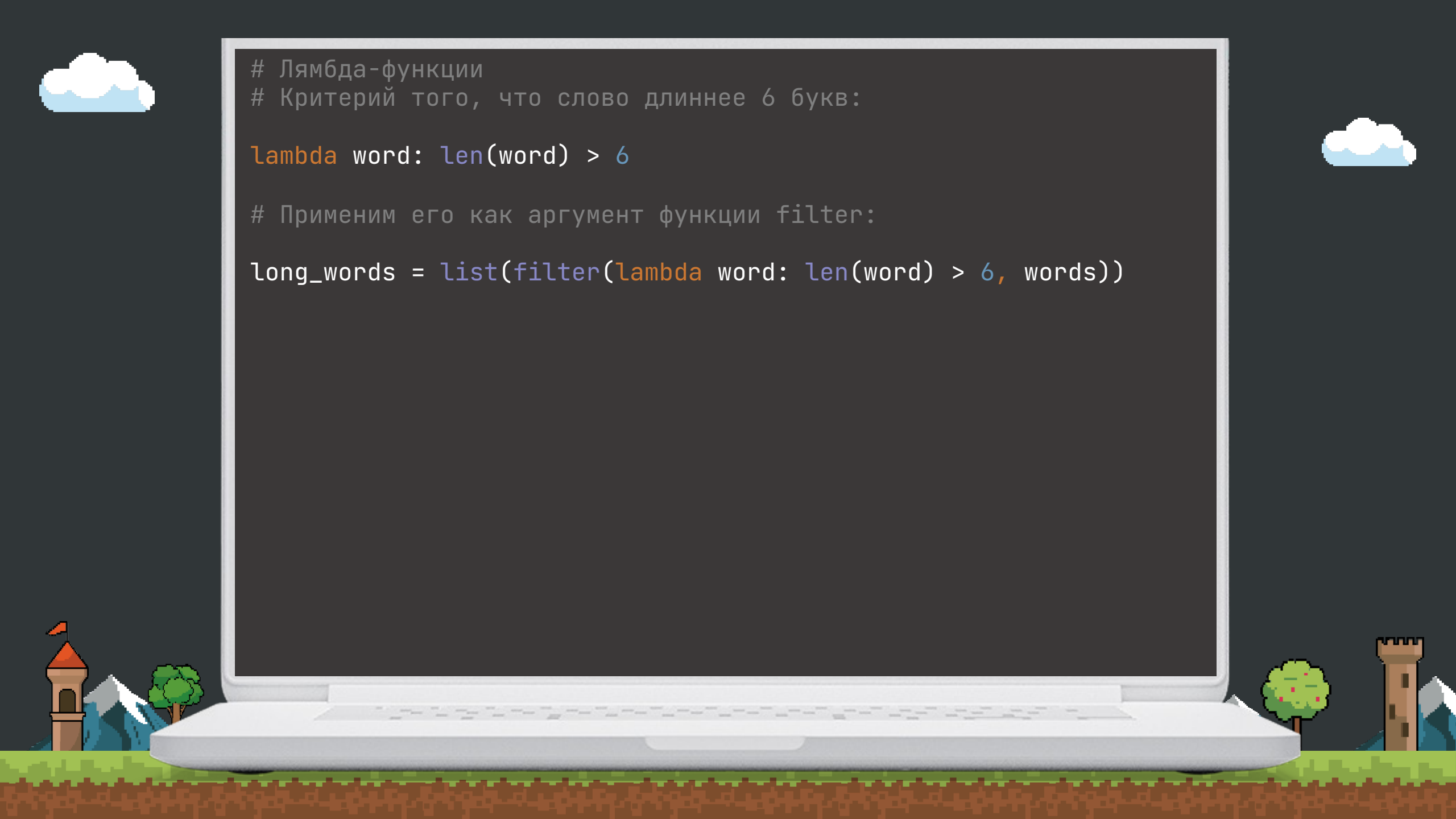




```
# Лямбда-функции
# Часто требуются очень простые функции,
# например Как критерий отбора.
# Они используются единожды и им не нужно даже имя

# Такие безымянные функции называются лямбда-функциями
# и создаются так:
```

```
lambda <аргументы>: <выражение>
```





```
# Лямбда-функции
# Критерий того, что слово длиннее 6 букв:

lambda word: len(word) > 6

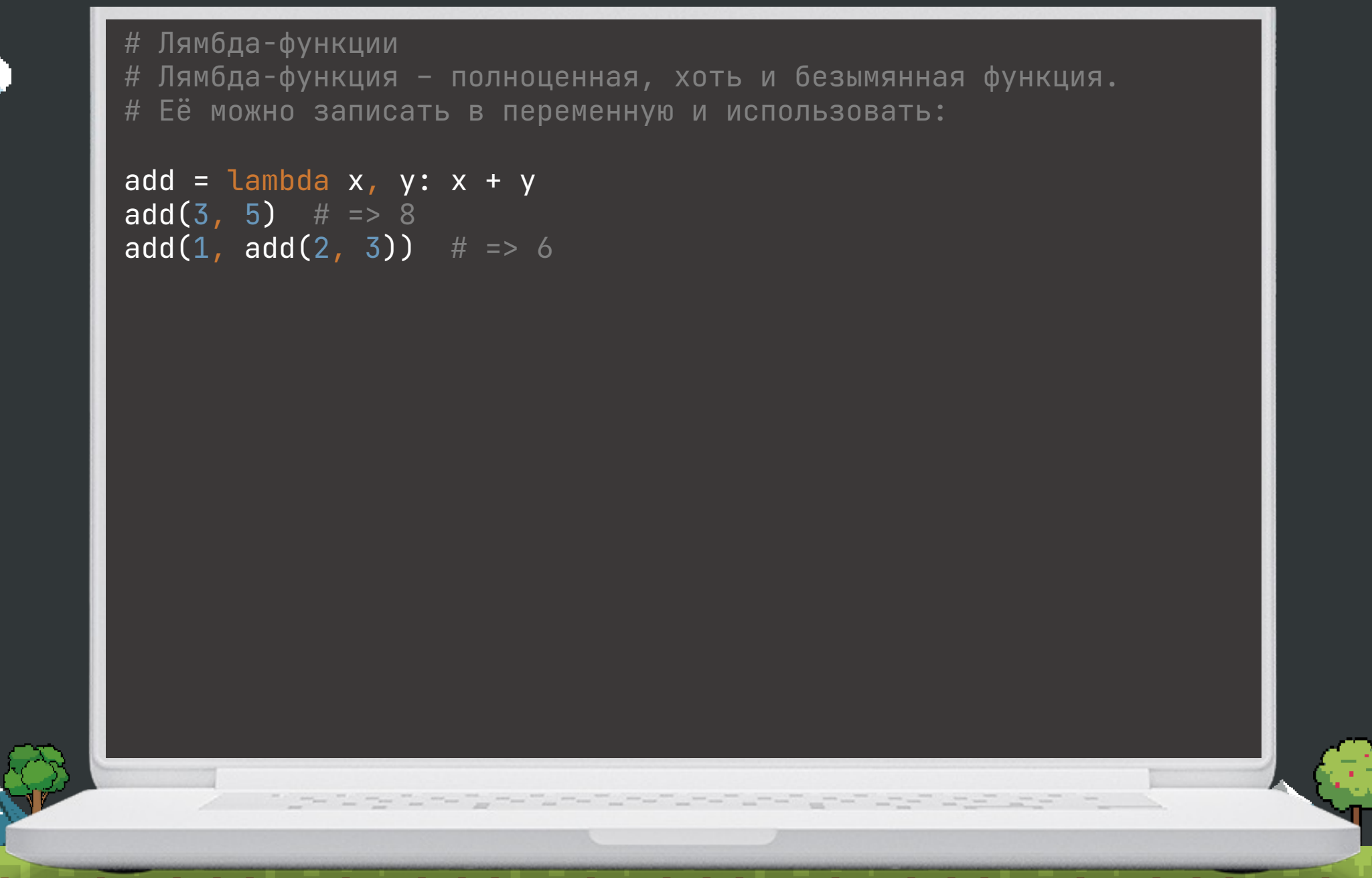
# Применим его как аргумент функции filter:

long_words = list(filter(lambda word: len(word) > 6, words))
```

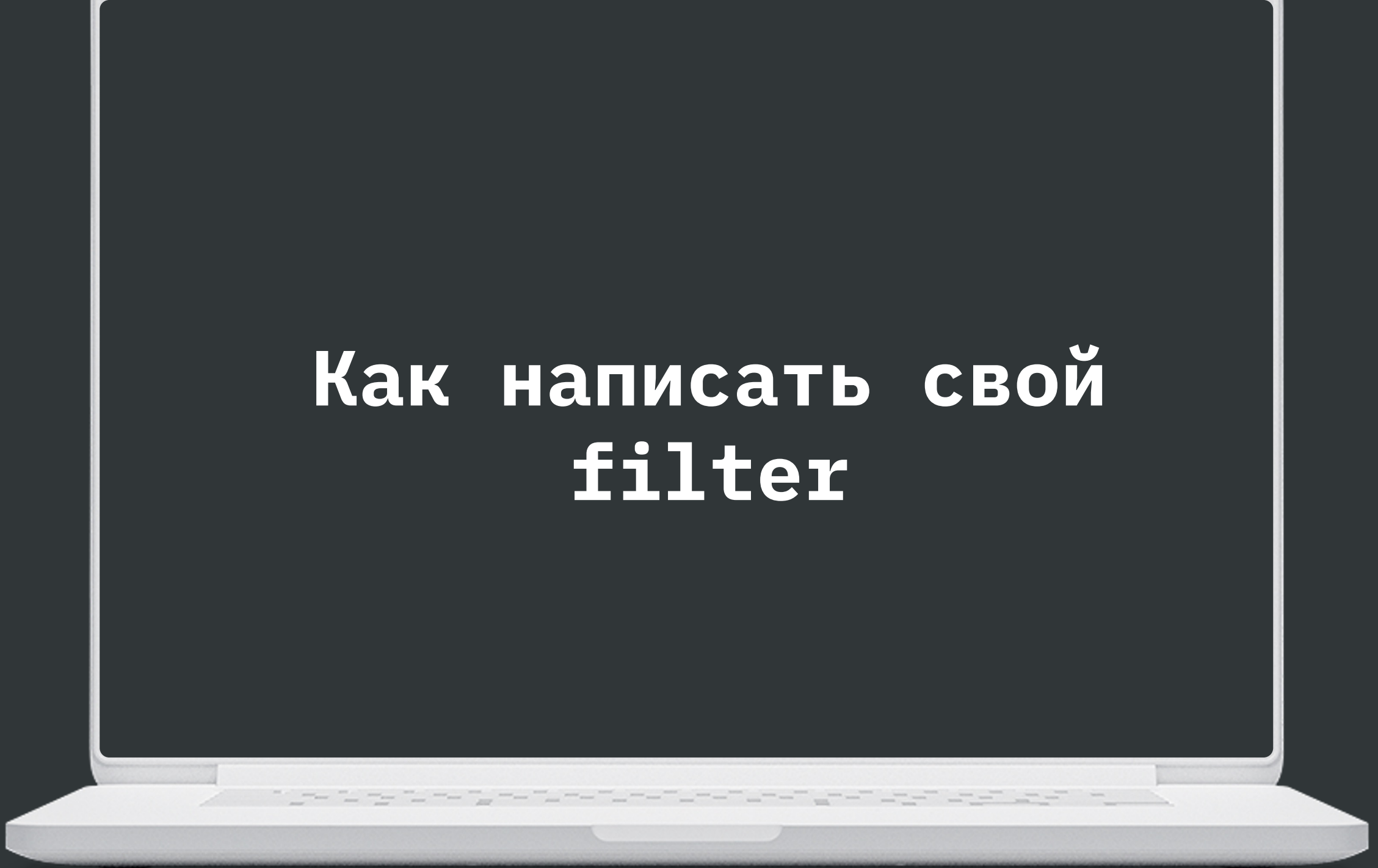


```
# Лямбда-функции
# Лямбда-функция – полноценная, хоть и безымянная функция.
# Её можно записать в переменную и использовать:
```

```
add = lambda x, y: x + y
add(3, 5) # => 8
add(1, add(2, 3)) # => 6
```



Как написать свой filter







```
# Убираем магию. Пишем свой filter
```

```
def simple_filter(criterion, arr):  
    result = []  
    for element in arr:  
        if criterion(element):  
            result.append(element)  
    return result
```

```
simple_filter(lambda x: x % 12 == 7, range(1, 100))  
# => [7, 19, 31, 43, 55, 67, 79, 91]
```

A white laptop is shown from a front-facing perspective, slightly angled. The screen is dark and displays the text 'Функция map' in a white, bold, sans-serif font. The laptop's keyboard and trackpad are visible at the bottom.

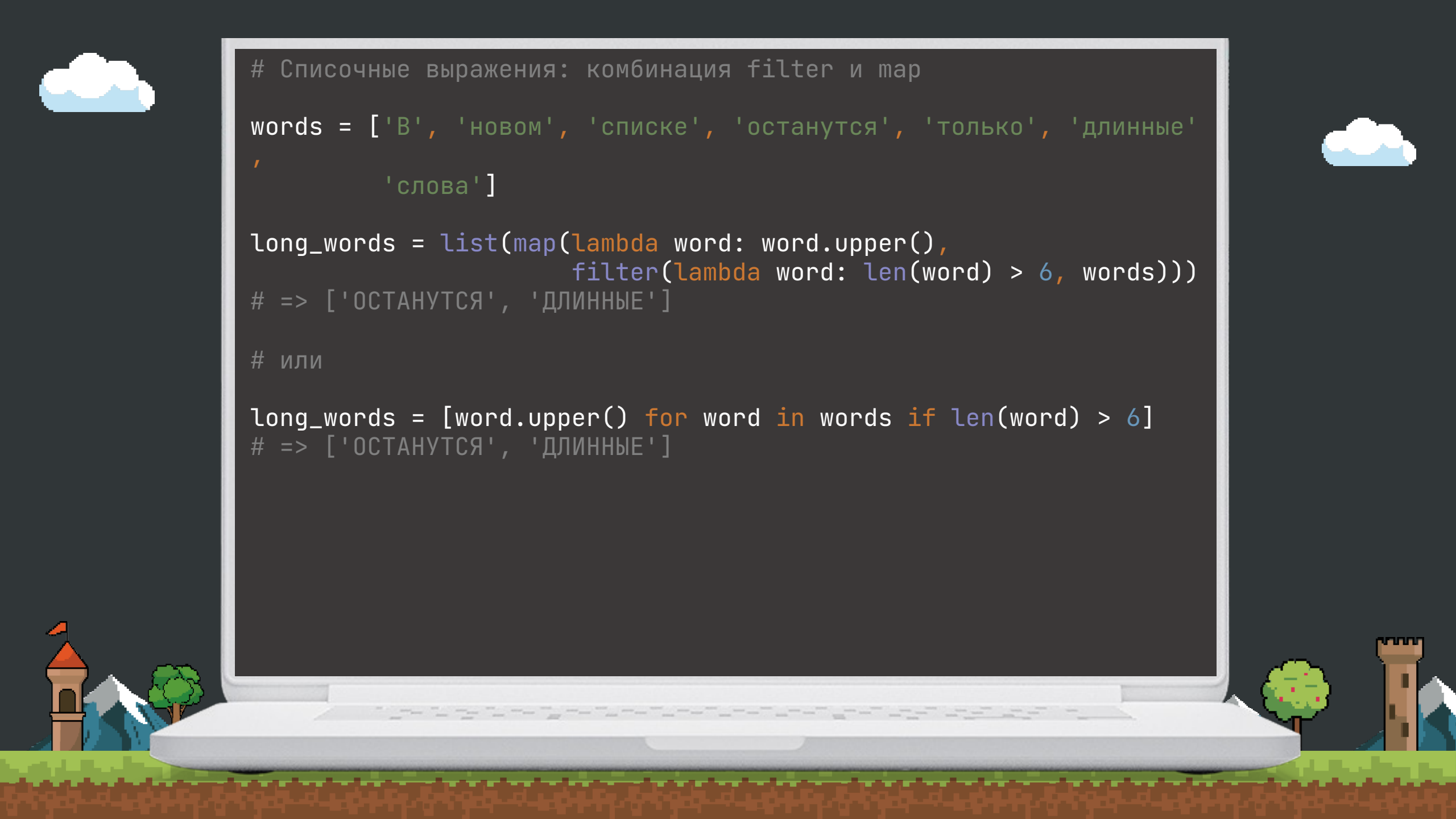
Функция map



```
# Функция map – преобразования списков  
# Функция map берёт функцию для преобразования одного элемента  
# и список. Выполняет преобразование всех элементов списка
```

```
list(map(lambda x: x ** 2, range(1, 10)))  
# => [1, 4, 9, 16, 25, 36, 49, 64, 81]
```





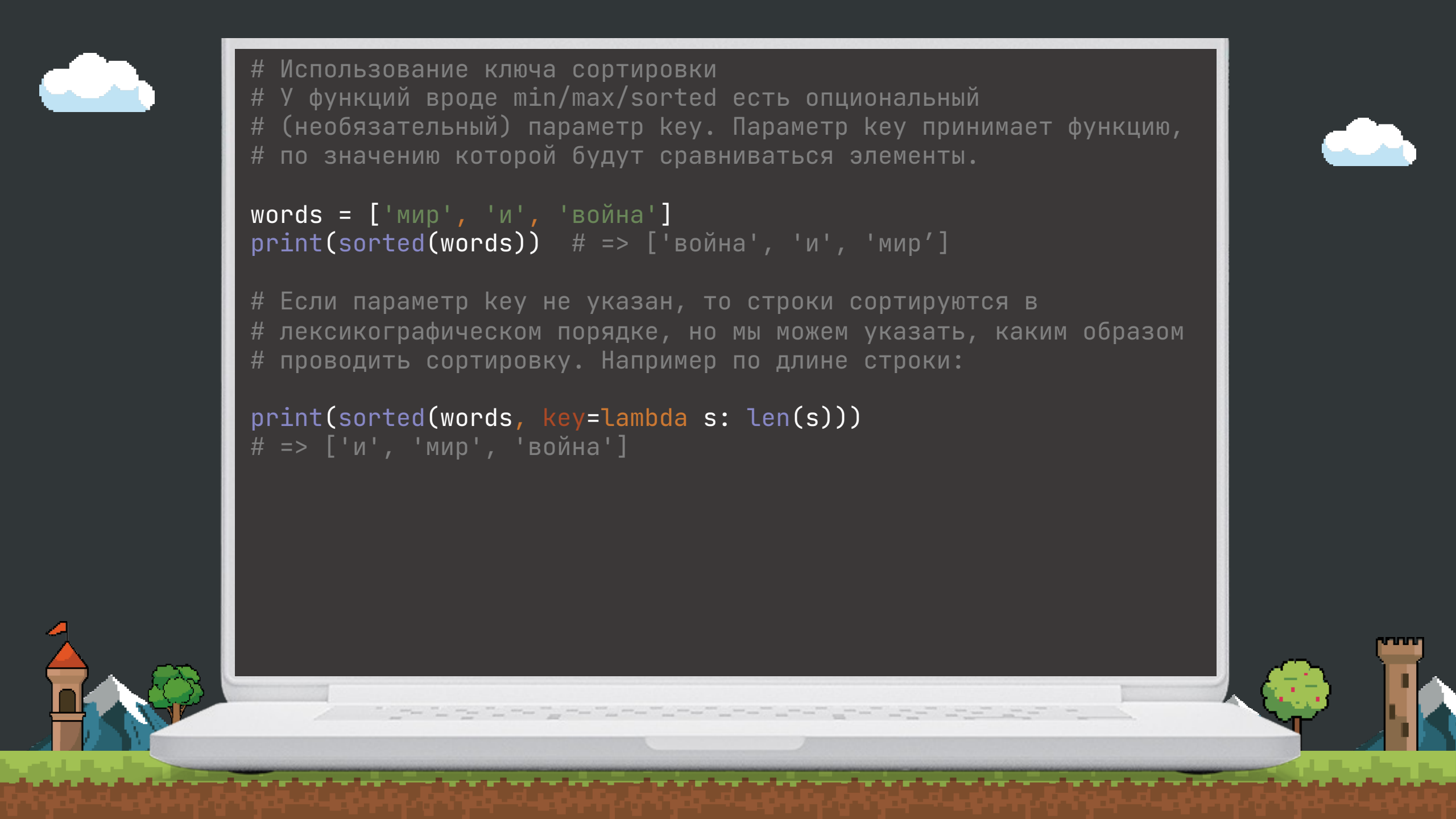
```
# Списочные выражения: комбинация filter и map

words = ['В', 'новом', 'списке', 'останутся', 'только', 'длинные',
         'слова']

long_words = list(map(lambda word: word.upper(),
                      filter(lambda word: len(word) > 6, words)))
# => ['ОСТАНУТСЯ', 'ДЛИННЫЕ']

# или

long_words = [word.upper() for word in words if len(word) > 6]
# => ['ОСТАНУТСЯ', 'ДЛИННЫЕ']
```



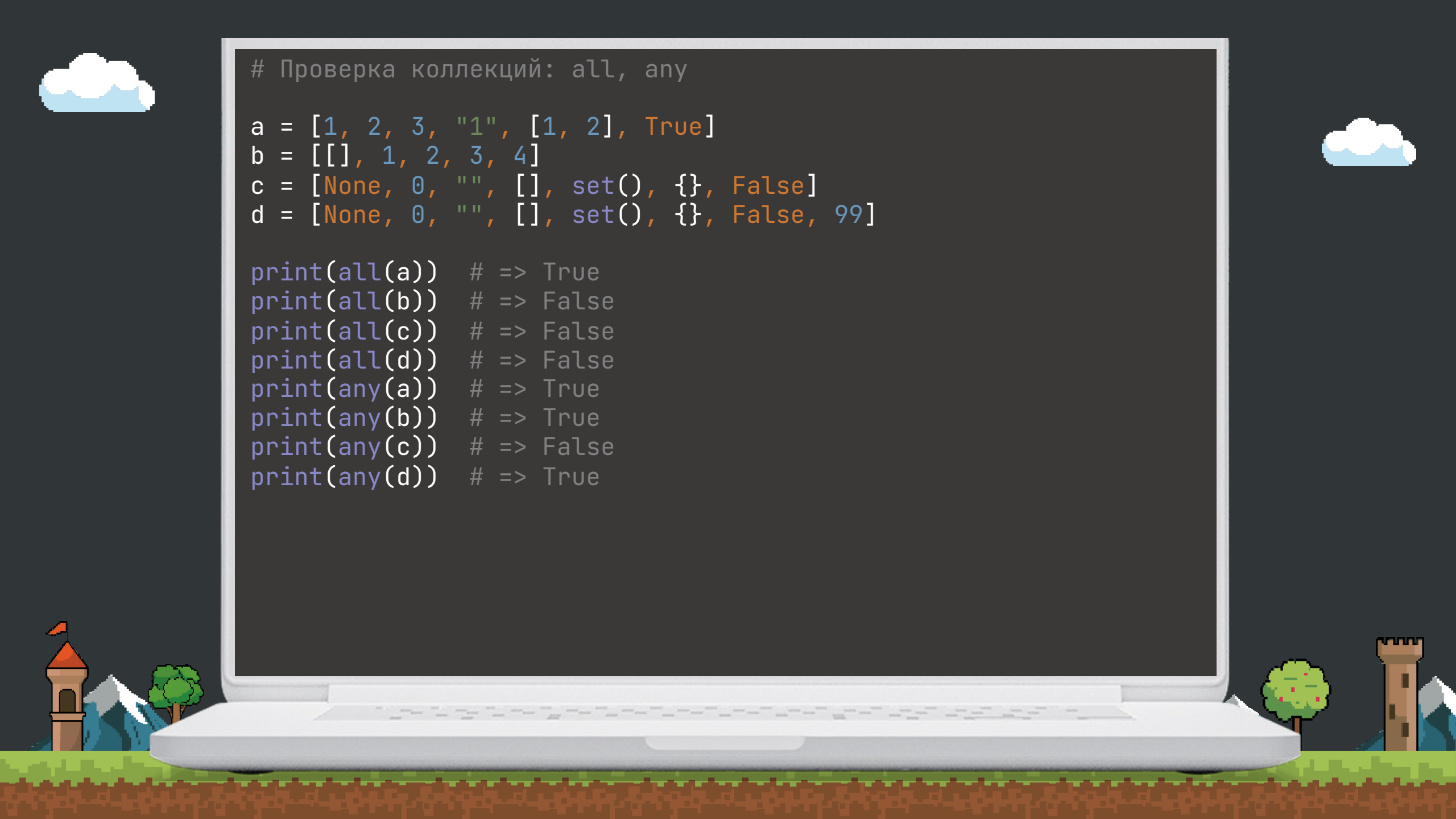
```
# Использование ключа сортировки
# У функций вроде min/max/sorted есть опциональный
# (необязательный) параметр key. Параметр key принимает функцию,
# по значению которой будут сравниваться элементы.
```

```
words = ['мир', 'и', 'война']
print(sorted(words)) # => ['война', 'и', 'мир']
```

```
# Если параметр key не указан, то строки сортируются в
# лексикографическом порядке, но мы можем указать, каким образом
# проводить сортировку. Например по длине строки:
```

```
print(sorted(words, key=lambda s: len(s)))
# => ['и', 'мир', 'война']
```

**Проверка коллекций: all,
any**



```
# Проверка коллекций: all, any
```

```
a = [1, 2, 3, "1", [1, 2], True]
```

```
b = [[], 1, 2, 3, 4]
```

```
c = [None, 0, "", [], set(), {}, False]
```

```
d = [None, 0, "", [], set(), {}, False, 99]
```

```
print(all(a))    # => True
```

```
print(all(b))    # => False
```

```
print(all(c))    # => False
```

```
print(all(d))    # => False
```

```
print(any(a))    # => True
```

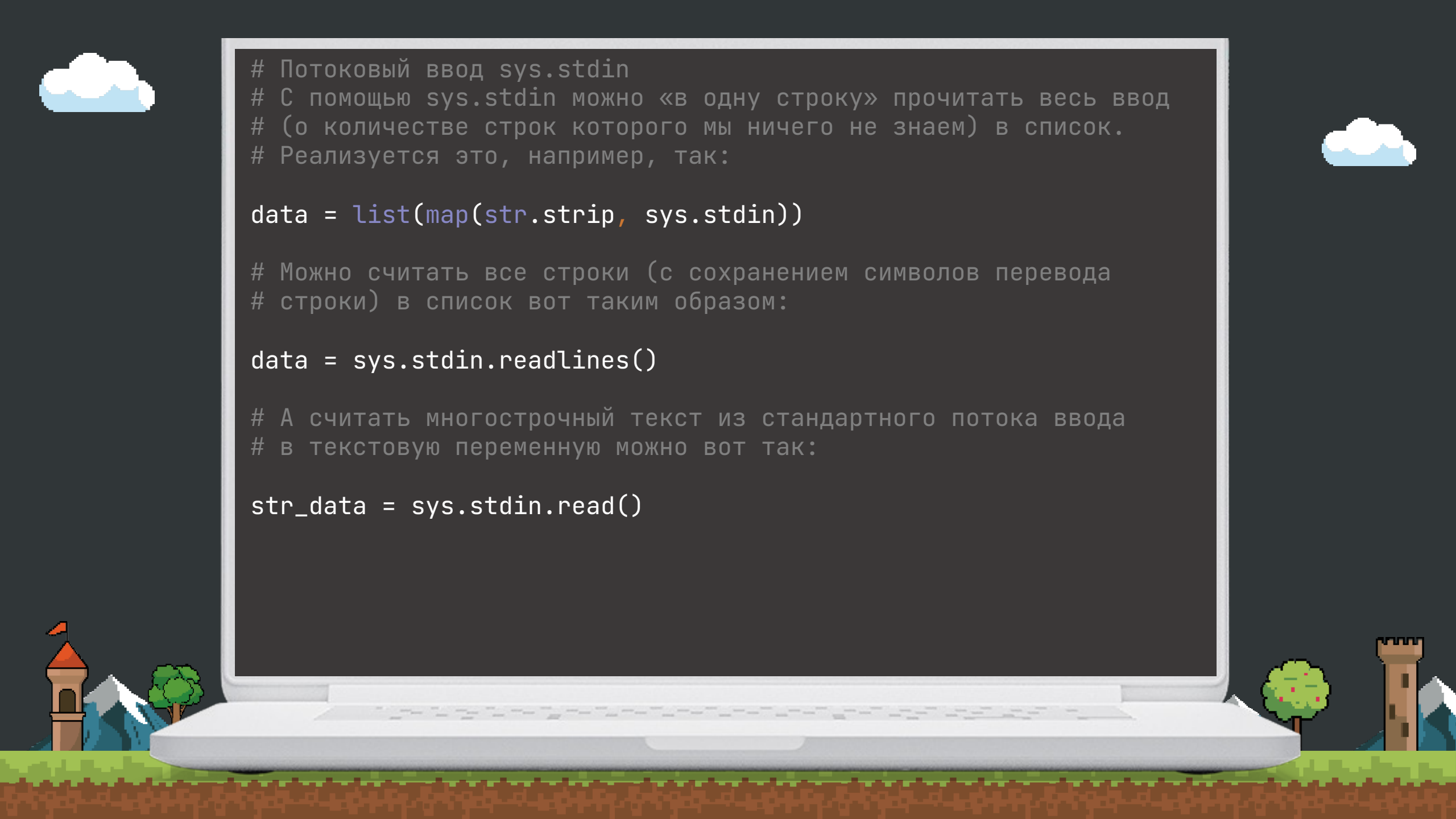
```
print(any(b))    # => True
```

```
print(any(c))    # => False
```

```
print(any(d))    # => True
```


A white laptop is shown from a front-facing perspective, slightly angled. The screen is dark and displays the text "ПОТОКОВЫЙ ВВОД stdin" in white. The keyboard area is visible at the bottom of the laptop frame.

ПОТОКОВЫЙ ВВОД `stdin`



```
# Поточковый ввод sys.stdin
# С помощью sys.stdin можно «в одну строку» прочитать весь ввод
# (о количестве строк которого мы ничего не знаем) в список.
# Реализуется это, например, так:
```

```
data = list(map(str.strip, sys.stdin))
```

```
# Можно считать все строки (с сохранением символов перевода
# строки) в список вот таким образом:
```

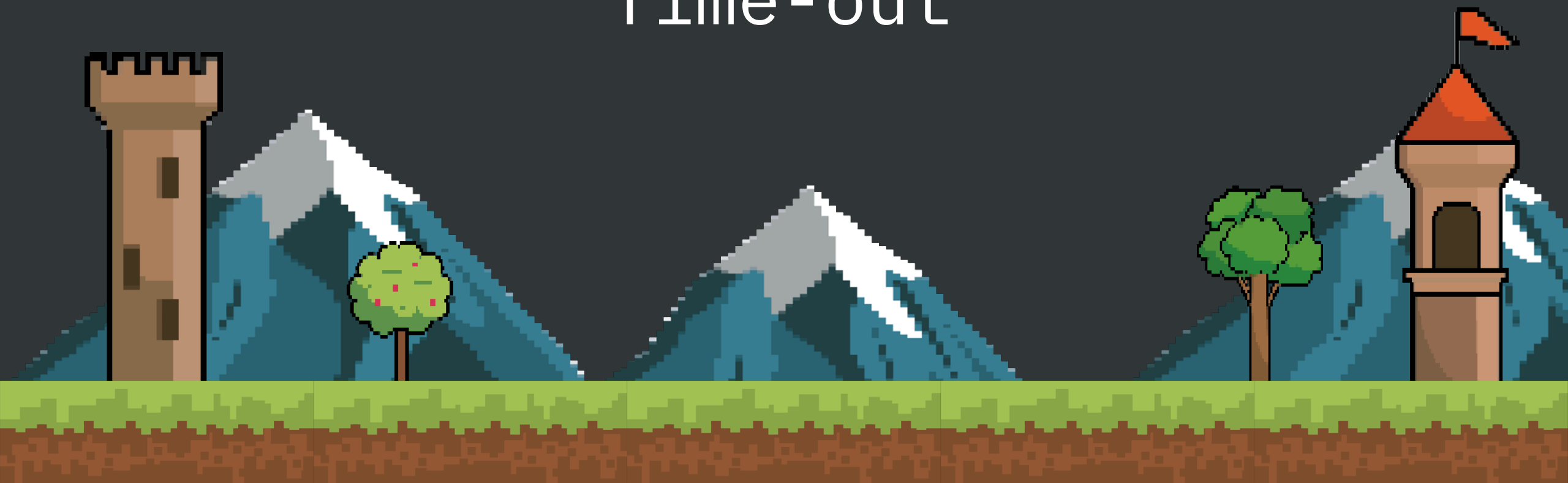
```
data = sys.stdin.readlines()
```

```
# А считать многострочный текст из стандартного потока ввода
# в текстовую переменную можно вот так:
```

```
str_data = sys.stdin.read()
```



Time-out





```
from memory_profiler import memory_usage

def list_func(size):
    return sum([1 for x in range(size) if x % 2 == 0])

def iter_func(size):
    return sum(1 for _ in filter(lambda x: x % 2 == 0, range(size)))

if __name__ == '__main__':
    size = 10 ** 9 # 1 000 000 000
    usage_list = memory_usage((list_func, (size,)))
    usage_iter = memory_usage((iter_func, (size,)))
    print(f'Max used memory LIST: {round(max(usage_list))}Mb')
    print(f'Max used memory ITER: {round(max(usage_iter))}Mb')
```

Max used memory LIST: 3870Mb

Max used memory ITER: 56Mb

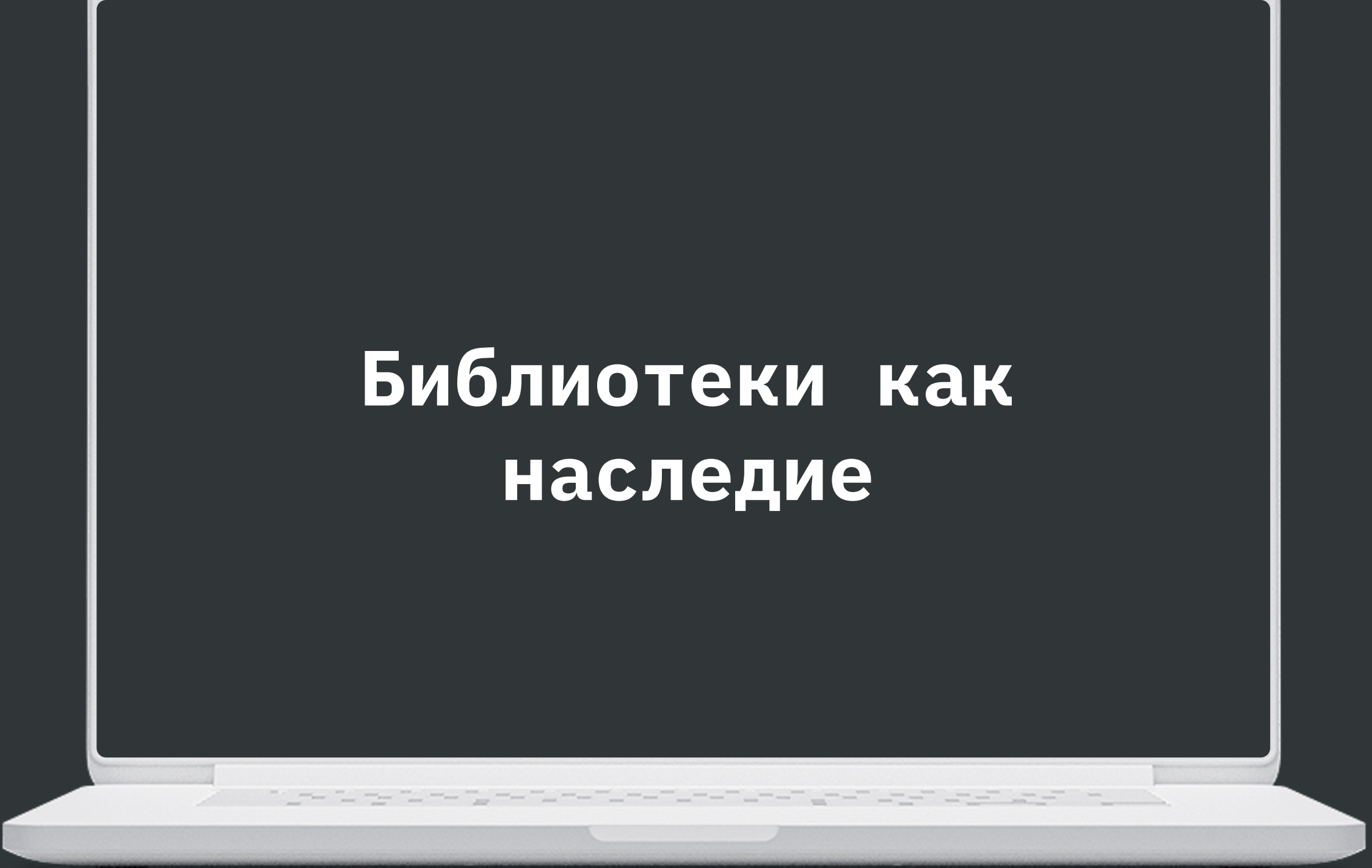




```
# enumerate
```

```
arr = ['This', 'is', 'third', ' word']  
print([pair for pair in enumerate(arr)])
```

```
[(0, 'This'), (1, 'is'), (2, 'third'), (3, ' word')]
```

A white laptop is shown from a front-facing perspective, slightly angled. The screen is dark and displays the text 'Библиотеки как наследие' in a bold, white, sans-serif font. The laptop's keyboard and trackpad are visible at the bottom.

Библиотеки как наследие

https://pypi.org/

365,795 projects


3,337,276 releases


5,808,258 files

581,815 users

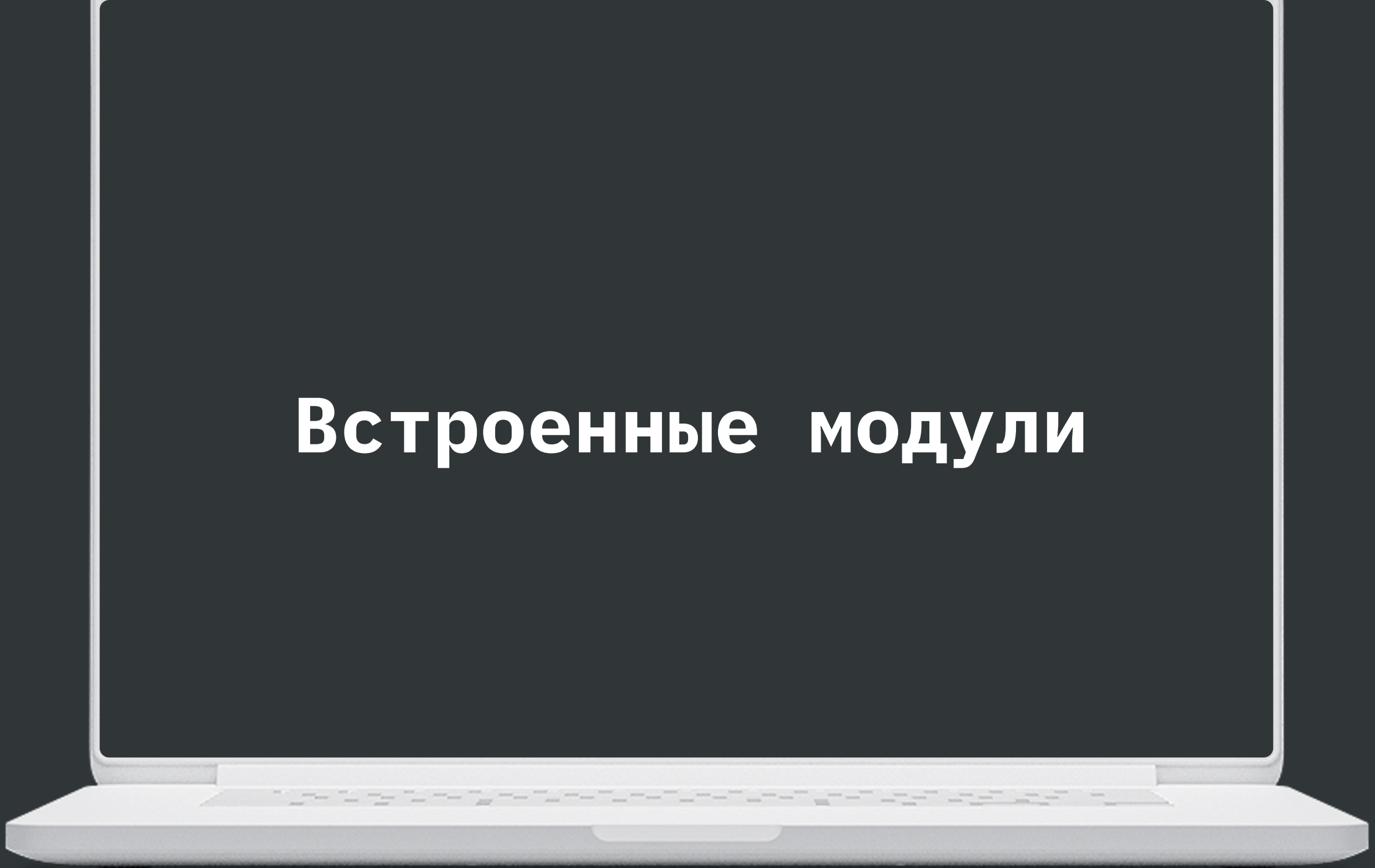


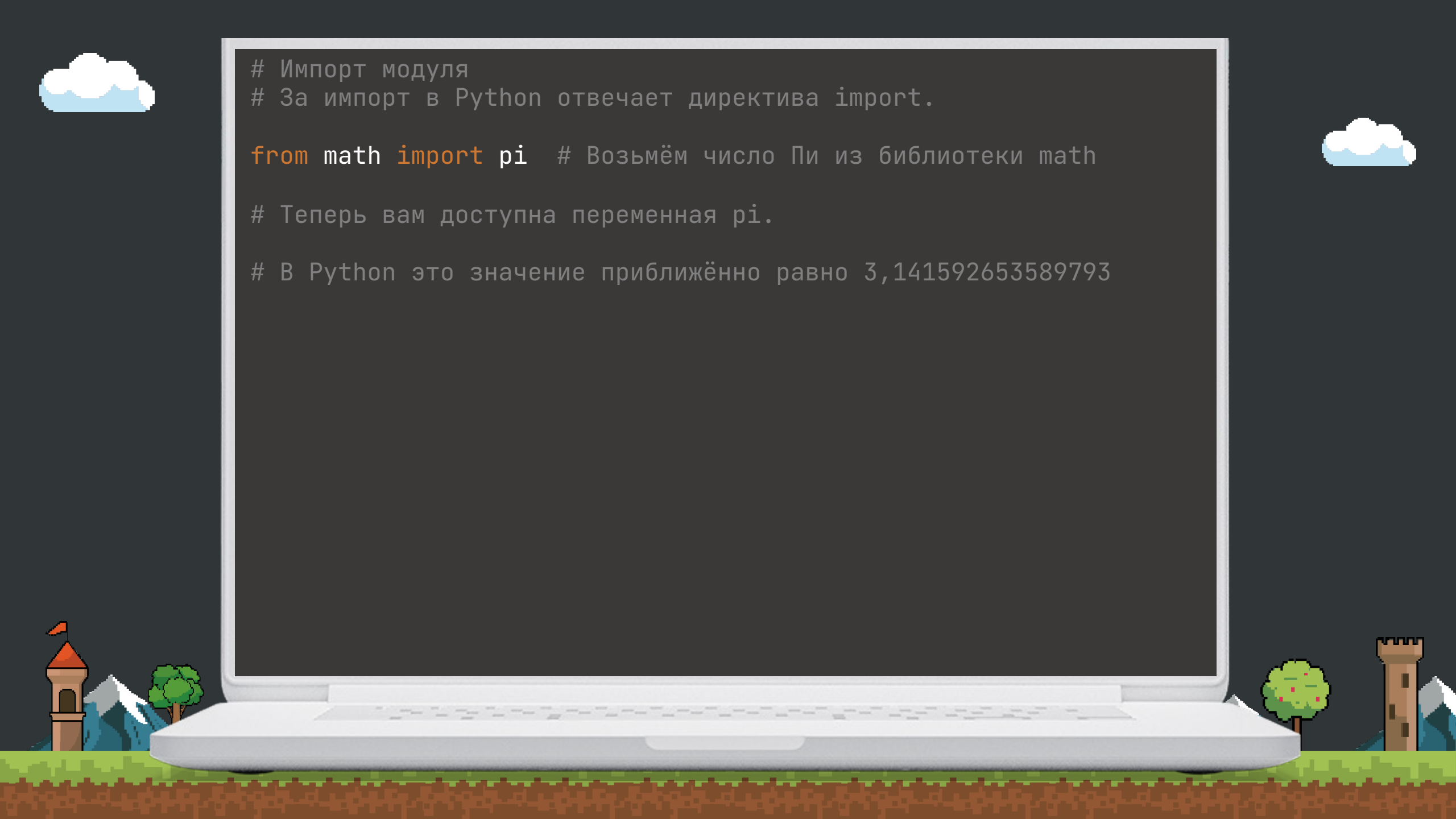
The Python Package Index (PyPI) is a repository of software for the Python programming language.

PyPI helps you find and install software developed and shared by the Python community. [Learn about installing packages](#) .

Package authors use PyPI to distribute their software. [Learn how to package your Python code for PyPI](#) .

Встроенные модули



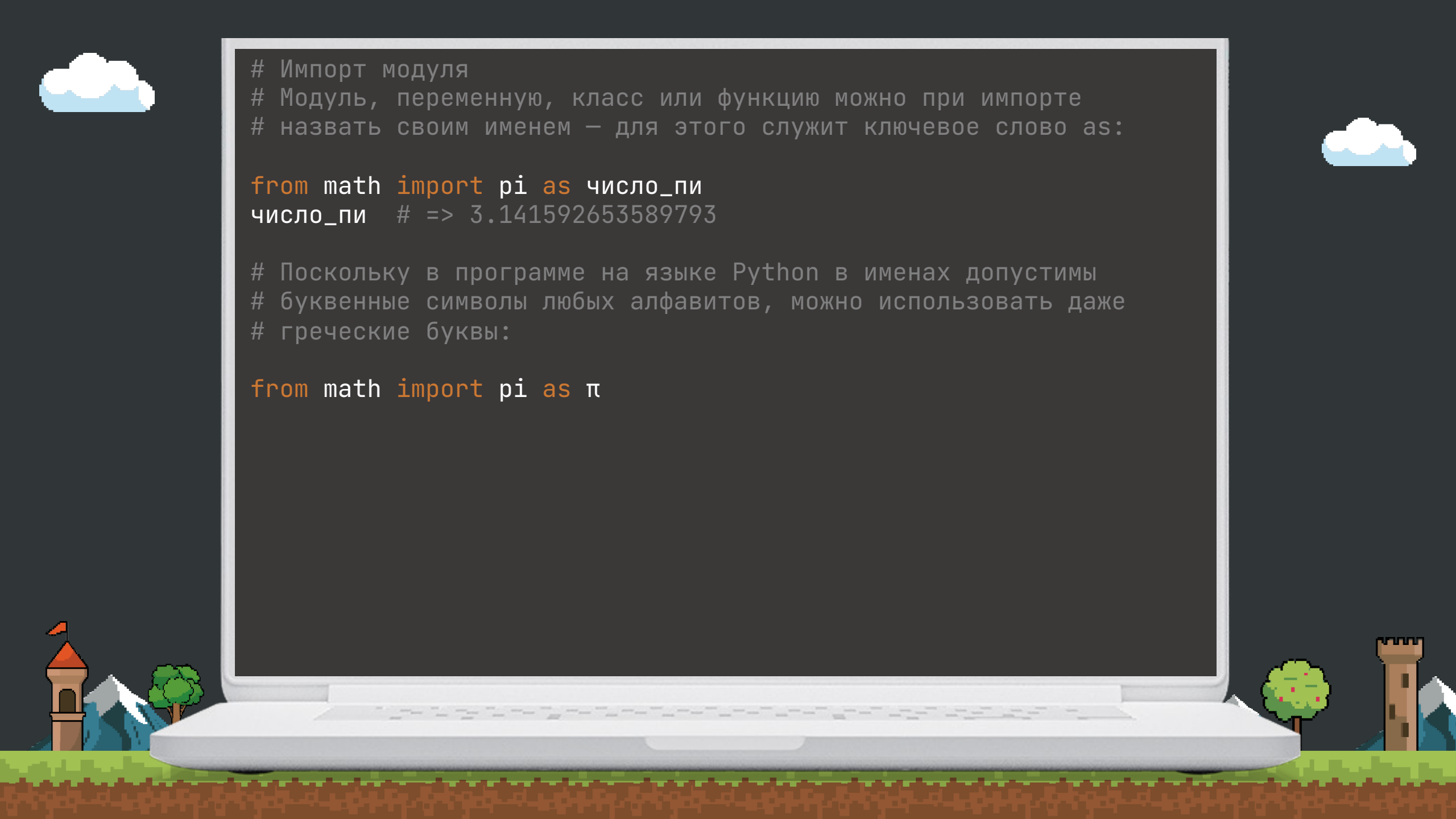


```
# Импорт модуля
# За импорт в Python отвечает директива import.

from math import pi # Возьмём число Пи из библиотеки math

# Теперь вам доступна переменная pi.

# В Python это значение приближённо равно 3,141592653589793
```



```
# Импорт модуля
# Модуль, переменную, класс или функцию можно при импорте
# назвать своим именем — для этого служит ключевое слово as:
```

```
from math import pi as число_пи
число_пи # => 3.141592653589793
```

```
# Поскольку в программе на языке Python в именах допустимы
# буквенные символы любых алфавитов, можно использовать даже
# греческие буквы:
```

```
from math import pi as π
```



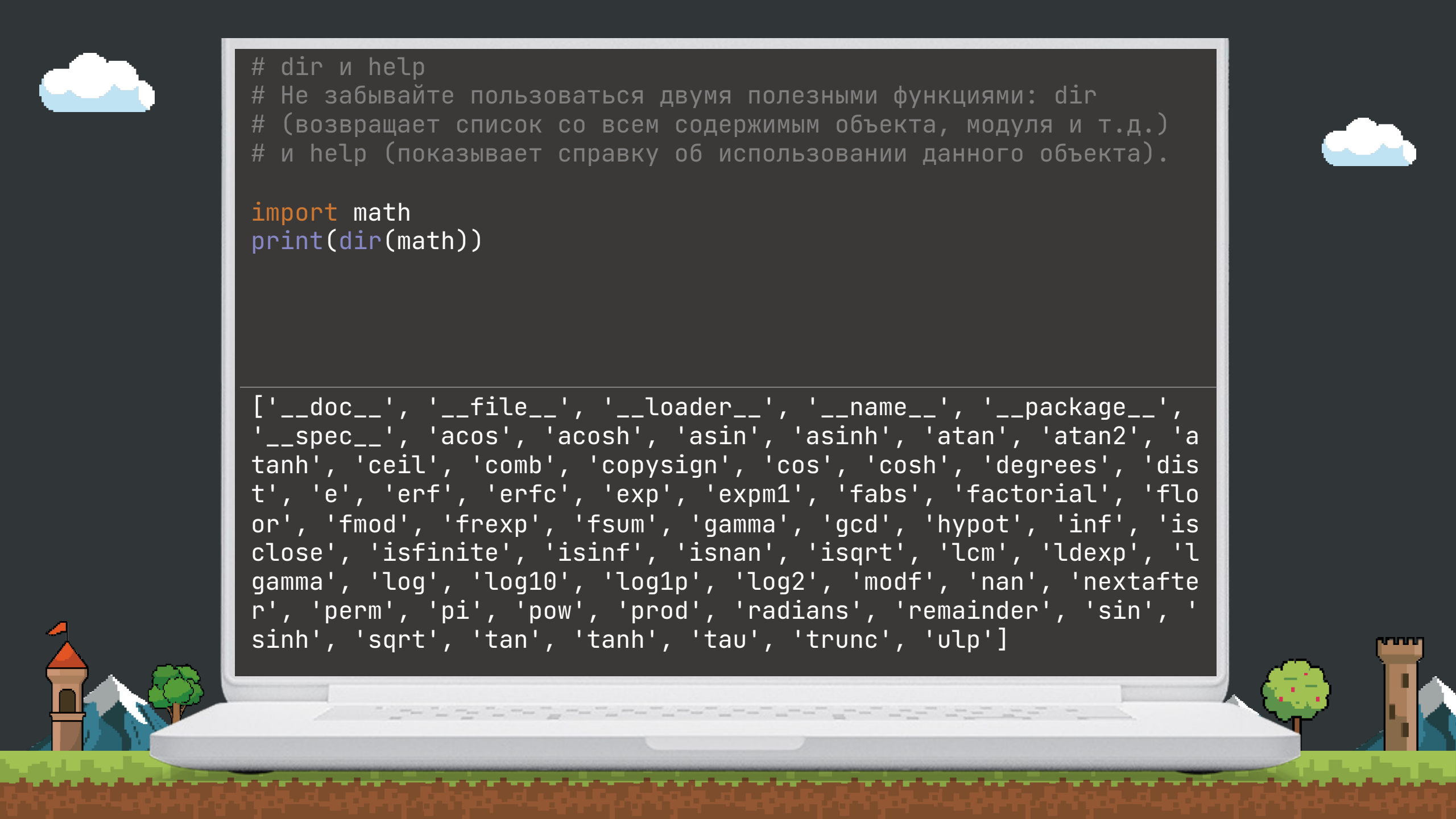
```
# Импорт модуля  
# Значения после директивы import можно писать через запятую:
```

```
from math import sin, cos, tan
```

```
# Значок «*» означает, что из библиотеки нужно импортировать  
# всё, что доступно:
```

```
from math import *
```

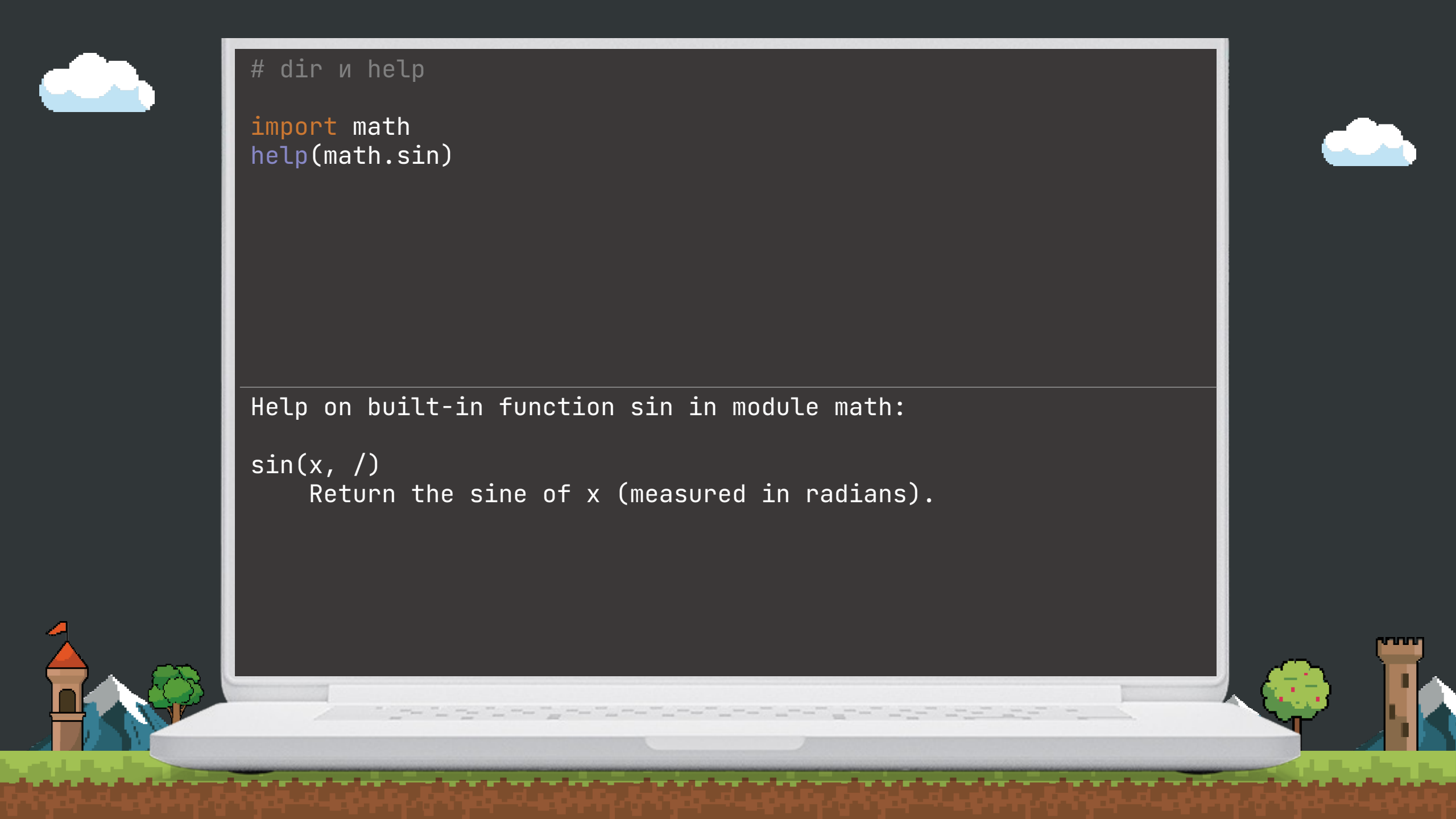
```
# Впрочем, так делать не рекомендуется, поскольку при таком  
# подходе засоряется пространство имён.
```



```
# dir и help
# Не забывайте пользоваться двумя полезными функциями: dir
# (возвращает список со всем содержимым объекта, модуля и т.д.)
# и help (показывает справку об использовании данного объекта).
```

```
import math
print(dir(math))
```

```
['__doc__', '__file__', '__loader__', '__name__', '__package__',
 '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'a
tanh', 'ceil', 'comb', 'copysign', 'cos', 'cosh', 'degrees', 'dis
t', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'flo
or', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'is
close', 'isfinite', 'isinf', 'isnan', 'isqrt', 'lcm', 'ldexp', 'l
gamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'nextafte
r', 'perm', 'pi', 'pow', 'prod', 'radians', 'remainder', 'sin', '
sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc', 'ulp']
```



```
# dir n help
```

```
import math  
help(math.sin)
```

Help on built-in function sin in module math:

```
sin(x, /)  
    Return the sine of x (measured in radians).
```

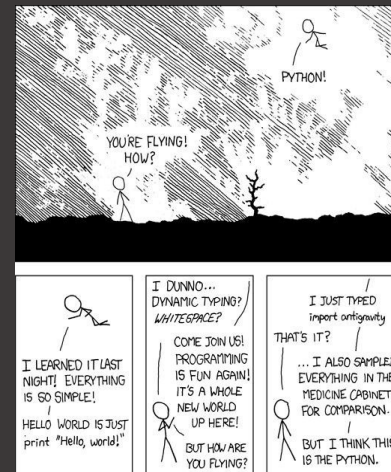
```
# «Пасхальные яйца»  
# Говоря про встроенную библиотеку, нельзя не сказать о «пасхальных  
# яйцах» в Python. При импорте модуля this вы познакомитесь с  
# дзенем Python.
```

```
import this
```

```
# А импорт модуля с антигравитацией откроет в браузере комикс  
# о том, что в Python действительно есть модули на все случаи жизни.
```

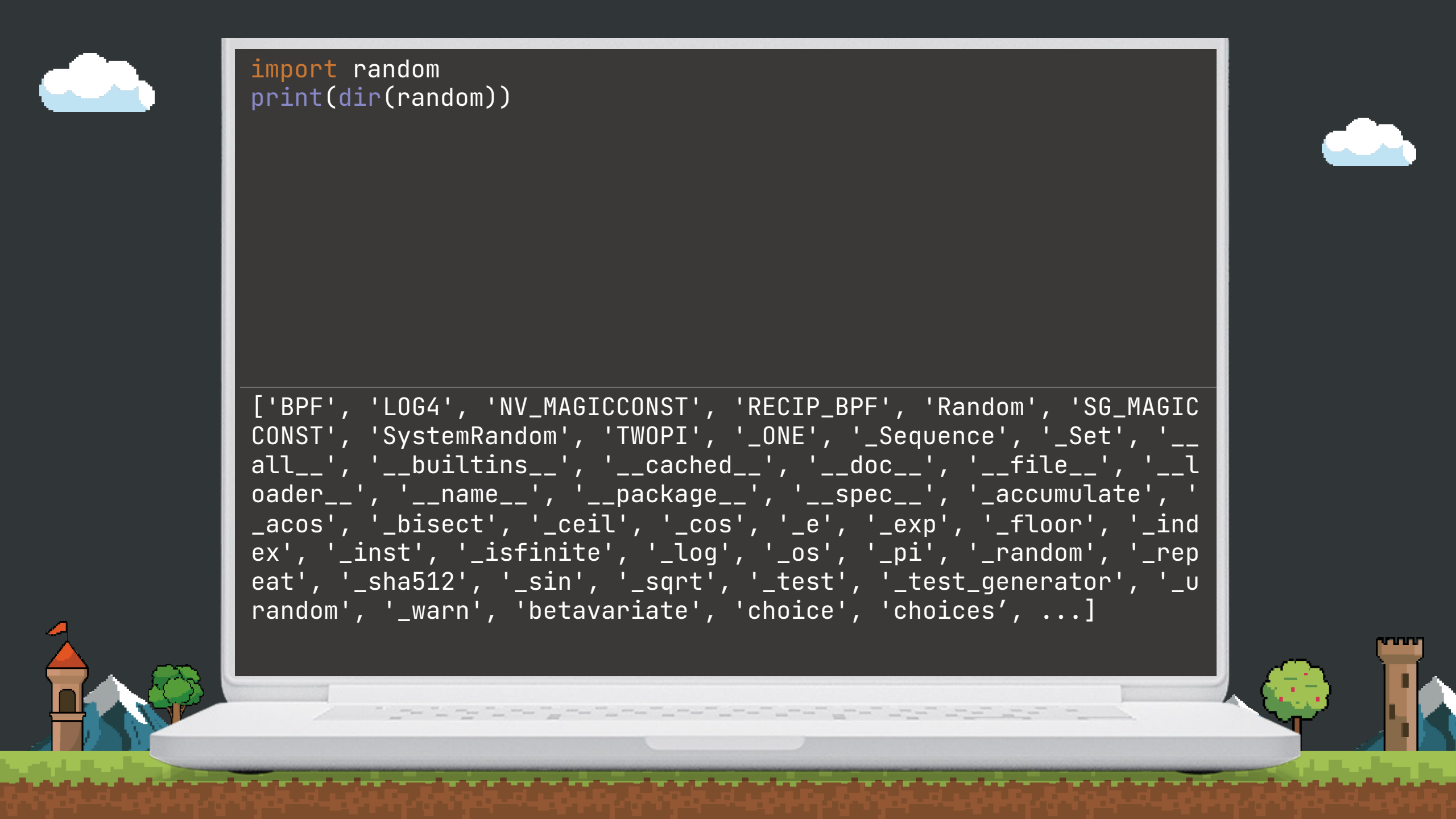
```
import antigravity
```

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
...



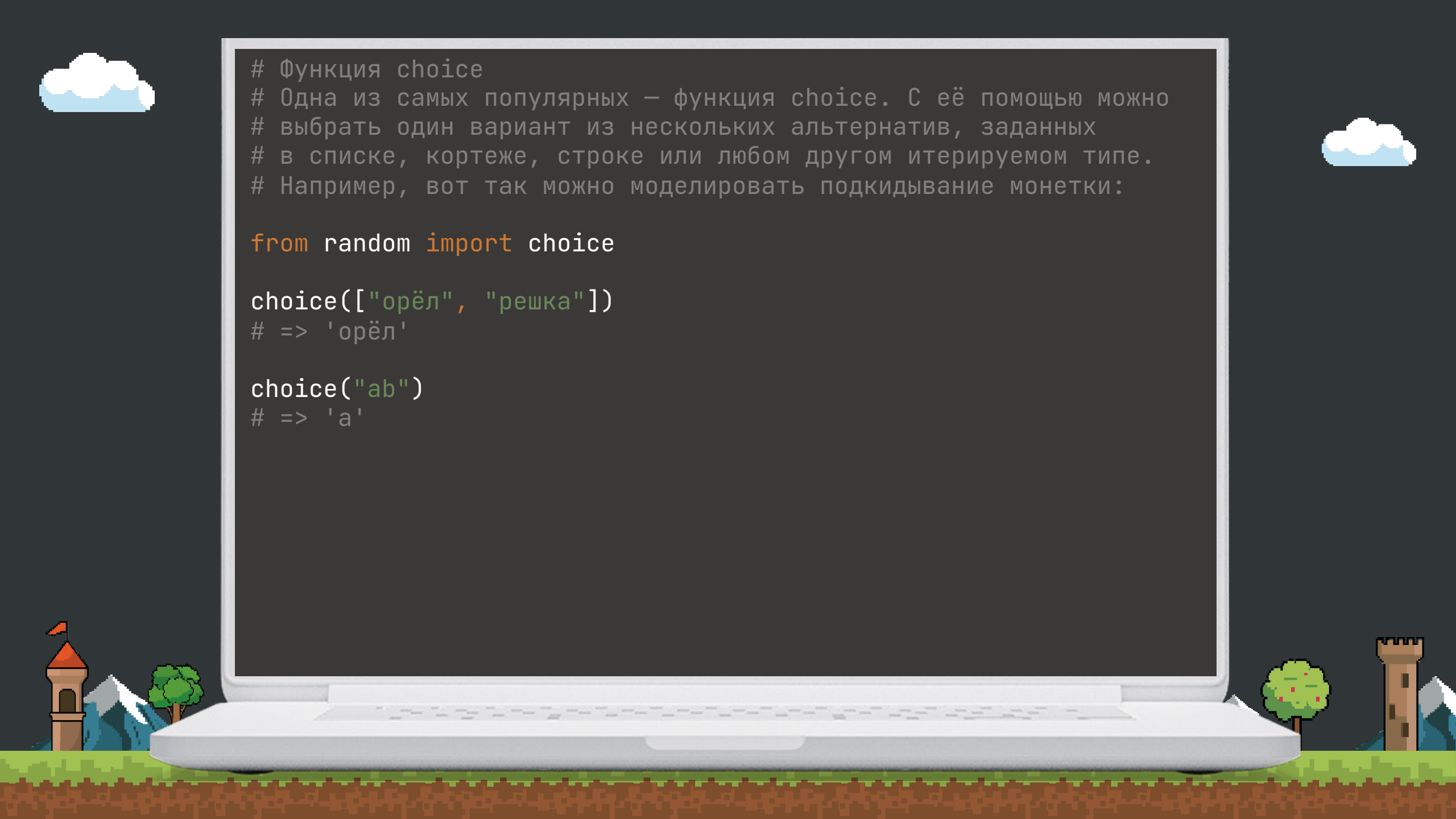
A white laptop is shown from a front-facing perspective, slightly angled. The screen is dark and displays the text 'Модуль random' in a white, bold, sans-serif font. The laptop's keyboard and trackpad are visible at the bottom.

Модуль random



```
import random
print(dir(random))
```

```
['BPF', 'LOG4', 'NV_MAGICCONST', 'RECIP_BPF', 'Random', 'SG_MAGICCONST', 'SystemRandom', 'TWOPI', '_ONE', '_Sequence', '_Set', '__all__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', '_accumulate', '_acos', '_bisect', '_ceil', '_cos', '_e', '_exp', '_floor', '_index', '_inst', '_isfinite', '_log', '_os', '_pi', '_random', '_repeat', '_sha512', '_sin', '_sqrt', '_test', '_test_generator', '_urandom', '_warn', 'betavariate', 'choice', 'choices', ...]
```

```
# Функция choice
# Одна из самых популярных — функция choice. С её помощью можно
# выбрать один вариант из нескольких альтернатив, заданных
# в списке, кортеже, строке или любом другом итерируемом типе.
# Например, вот так можно моделировать подкидывание монетки:
```

```
from random import choice
```

```
choice(["орёл", "решка"])
```

```
# => 'орёл'
```

```
choice("ab")
```

```
# => 'a'
```



```
# Функция choice  
# Можно симитировать несколько бросков игральных кубиков:
```

```
from random import choice
```

```
dashes = [1, 2, 3, 4, 5, 6]
```

```
for _ in range(1, 10):  
    print(choice(dashes), choice(dashes))
```

```
4 2  
2 5  
4 4  
6 2  
5 2  
5 6  
3 2  
5 5  
6 2
```





```
# Функция shuffle
```

```
from random import shuffle
```

```
a = list(range(100))  
shuffle(a) # меняет сам список  
print(a[:10])
```

```
[74, 28, 44, 9, 17, 99, 72, 7, 61, 62]
```

A white laptop is shown from a front-facing perspective, slightly angled. The screen is dark and displays the text 'Модуль datetime' in a white, bold, sans-serif font. The laptop's keyboard and trackpad are visible at the bottom.

Модуль datetime



```
# date
```

```
import datetime as dt
```

```
# тип данных 'дата' (год + месяц + день)
```

```
my_date = dt.date(2019, 11, 5)
```

```
print(my_date)
```

```
print(dt.date.today())
```

```
print(dt.date.today().weekday())
```

```
2019-11-05
```

```
2022-04-04
```

```
0
```

A white laptop is shown from a front-facing perspective, slightly angled. The screen is dark and displays the text 'Модуль pprint' in a white, bold, sans-serif font. The laptop's keyboard and trackpad are visible at the bottom of the frame.

Модуль pprint



```
# Модуль pprint
```

```
import random  
from pprint import pprint
```

```
a = [random.sample(range(20), 6) for _ in range(10)]  
pprint(a)
```

```
[[0, 3, 4, 18, 11, 15],  
 [18, 2, 8, 12, 6, 13],  
 [1, 16, 8, 6, 12, 2],  
 [9, 8, 4, 19, 15, 10],  
 [6, 2, 14, 19, 4, 18],  
 [0, 16, 4, 3, 12, 1],  
 [13, 8, 17, 19, 9, 14],  
 [6, 18, 9, 11, 15, 10],  
 [13, 8, 5, 10, 3, 17],  
 [1, 19, 10, 15, 4, 8]]
```

A white laptop is shown from a front-facing perspective, slightly angled. The screen is dark and displays the text 'Модуль itertools' in a white, bold, sans-serif font. The laptop's keyboard and trackpad are visible at the bottom of the frame.

Модуль `itertools`



```
# itertools.chain

from itertools import chain

chained = chain('ab', [33])
next(chained)  # a
next(chained)  # b
next(chained)  # 33
```



```
# Зацикливание: itertools.cycle, itertools.repeat
```

```
from itertools import repeat
```

```
repeater = repeat('PY')
```



```
next(repeater) # PY
```

```
next(repeater) # PY
```

```
next(repeater) # PY
```

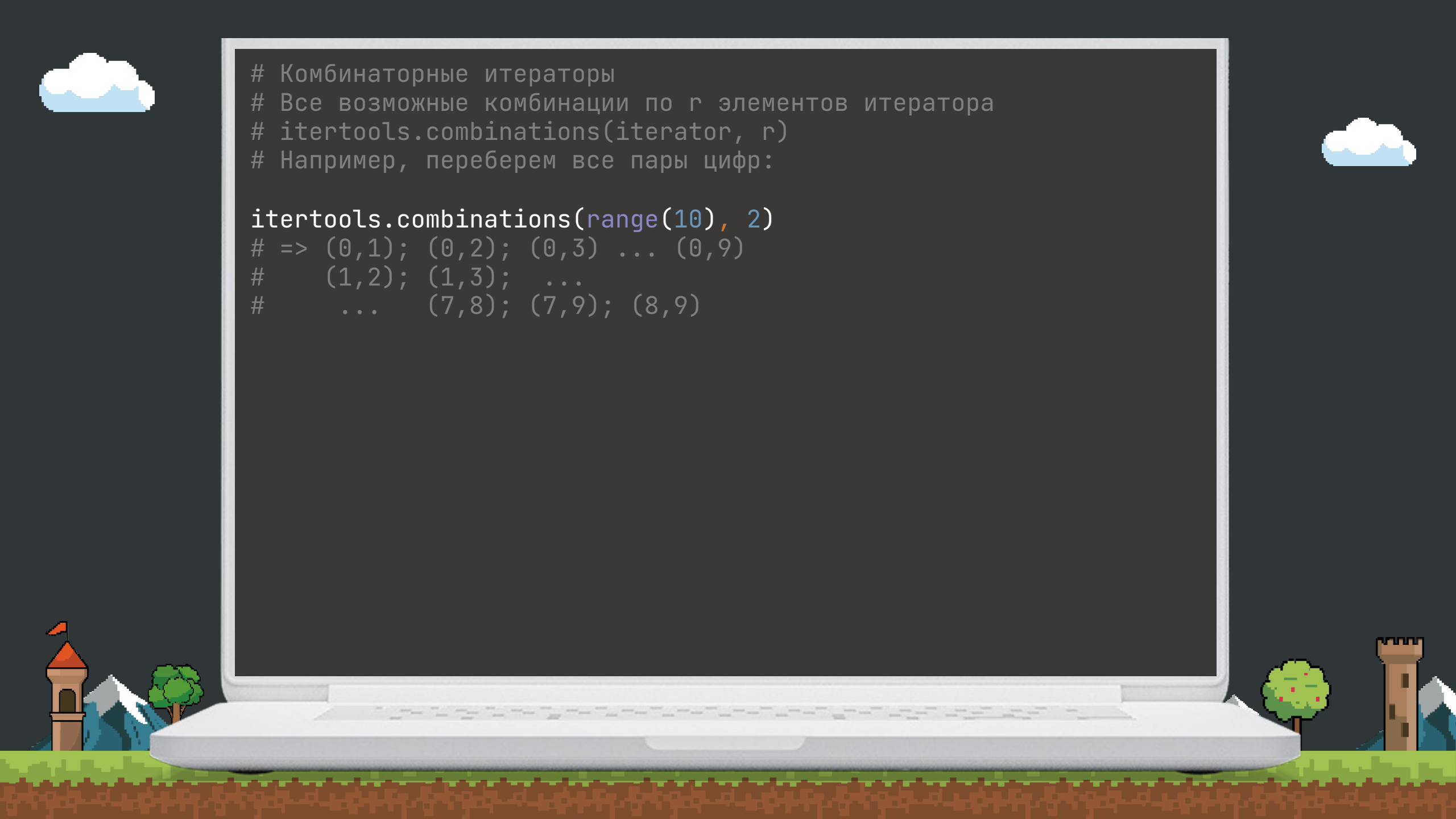
```
list(repeat('PY', 2)) # ['PY', 'PY']
```

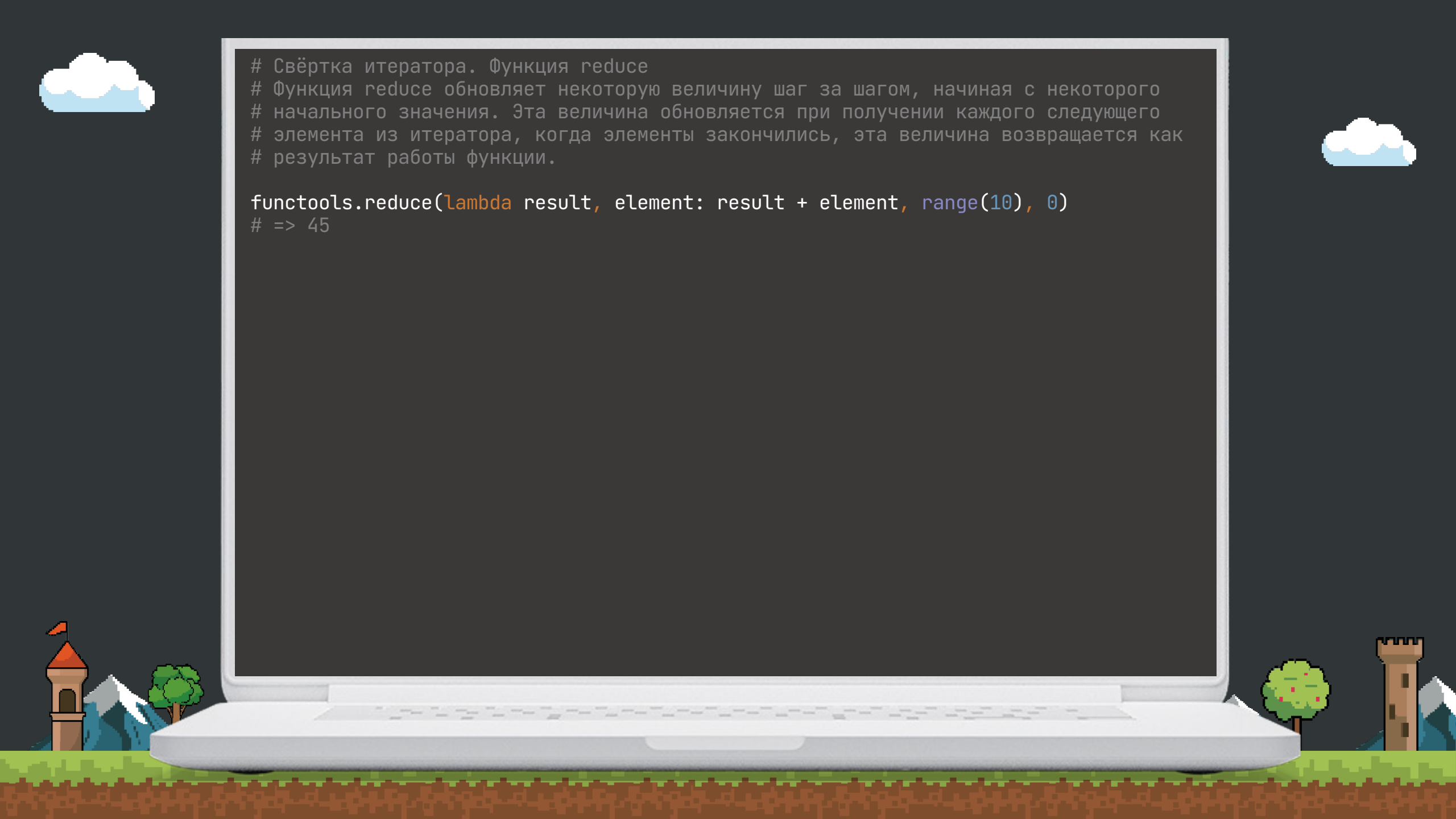




```
# Комбинаторные итераторы
# Все возможные комбинации по r элементов итератора
# itertools.combinations(iterator, r)
# Например, переберем все пары цифр:
```

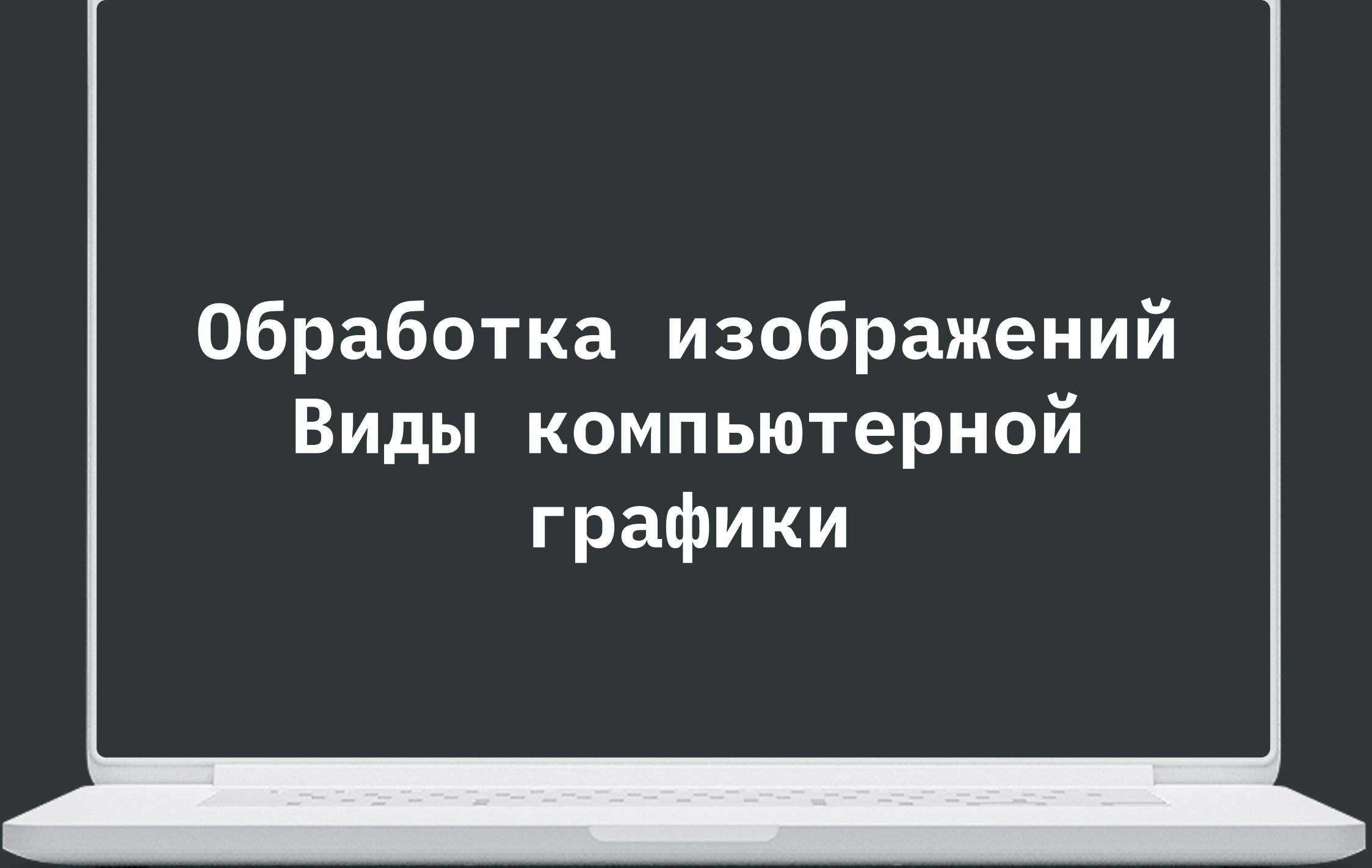
```
itertools.combinations(range(10), 2)
# => (0,1); (0,2); (0,3) ... (0,9)
#     (1,2); (1,3); ...
#     ... (7,8); (7,9); (8,9)
```





```
# Свёртка итератора. Функция reduce
# Функция reduce обновляет некоторую величину шаг за шагом, начиная с некоторого
# начального значения. Эта величина обновляется при получении каждого следующего
# элемента из итератора, когда элементы закончились, эта величина возвращается как
# результат работы функции.
```

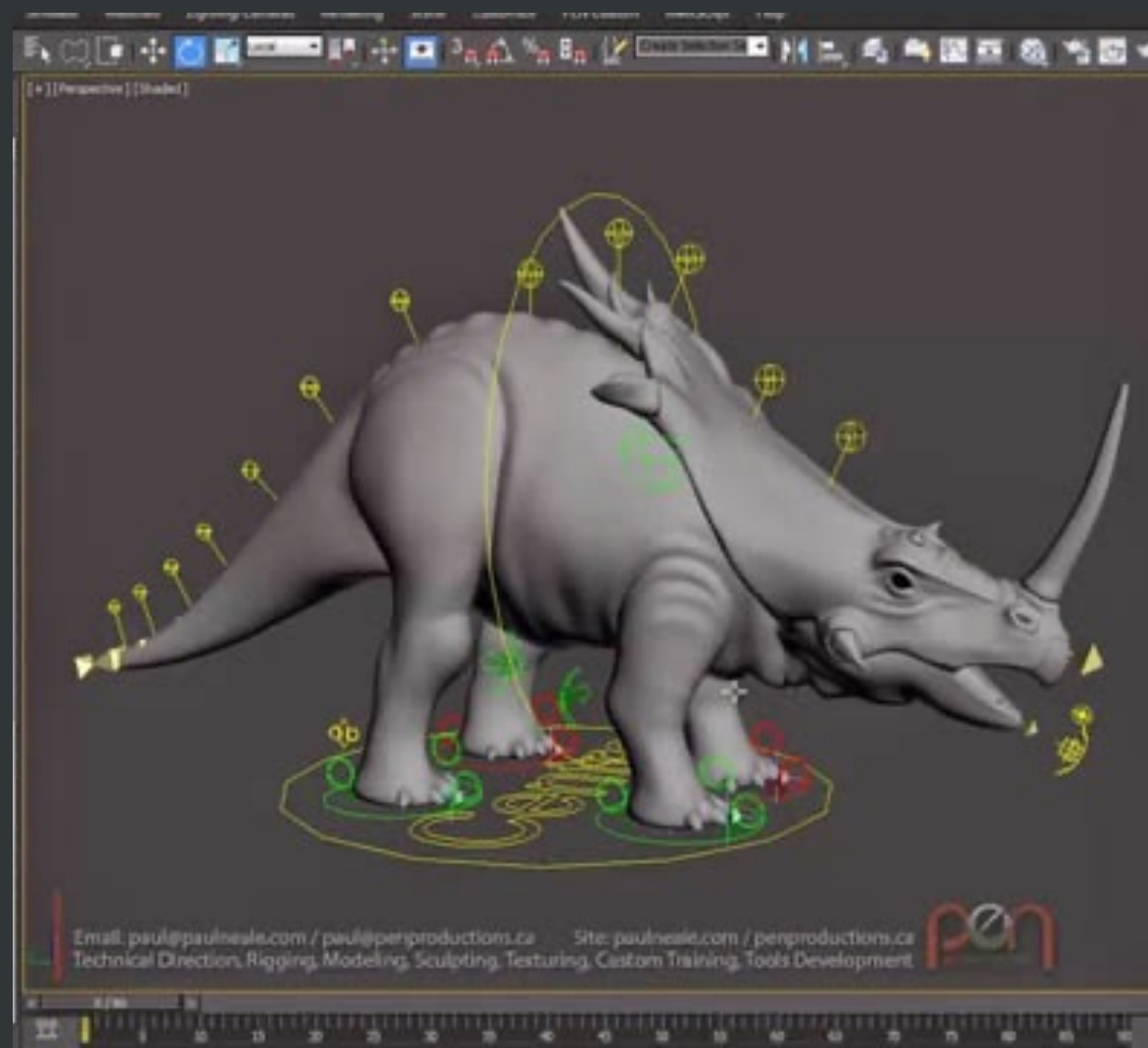
```
functools.reduce(lambda result, element: result + element, range(10), 0)
# => 45
```



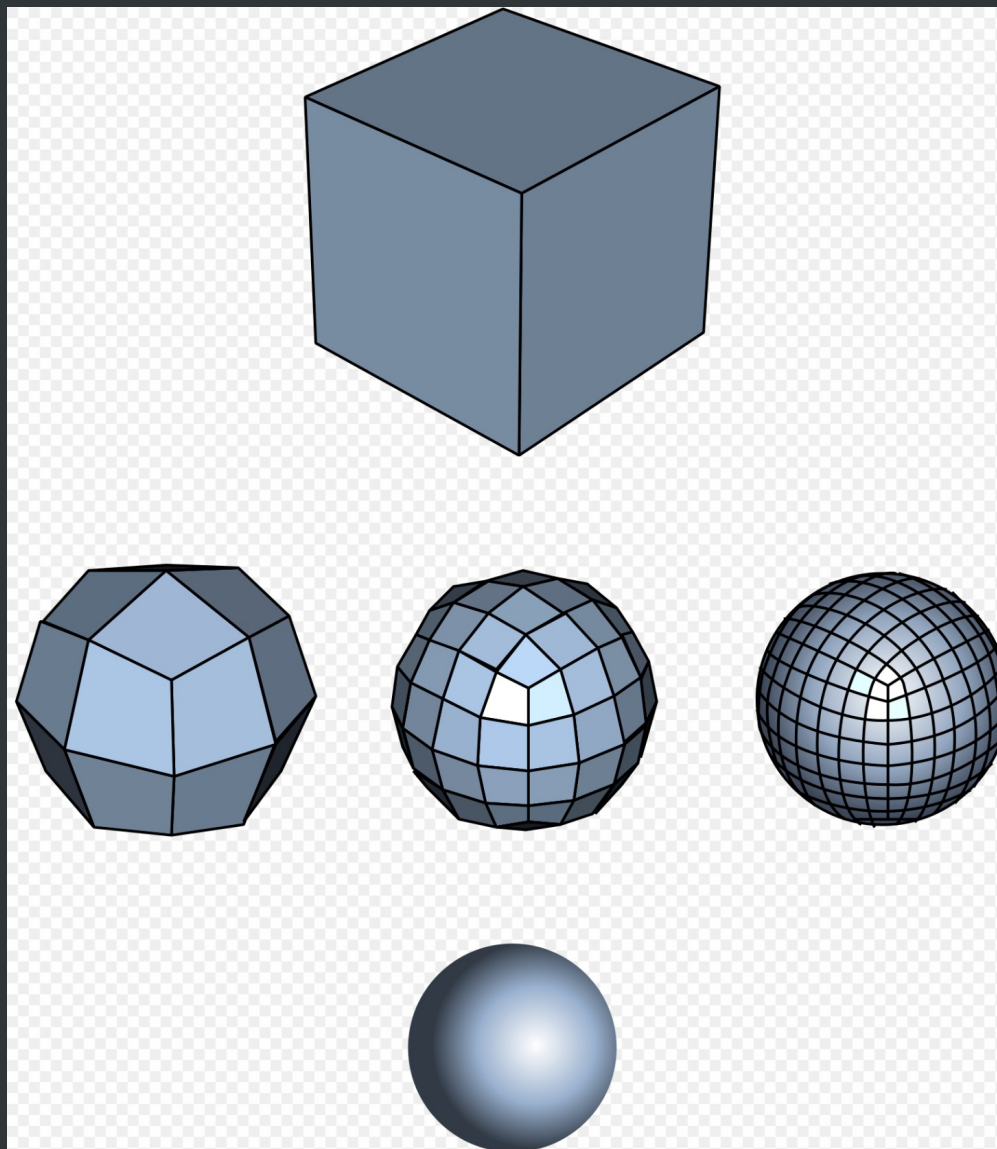
Обработка изображений

Виды компьютерной графики

Трёхмерная графика



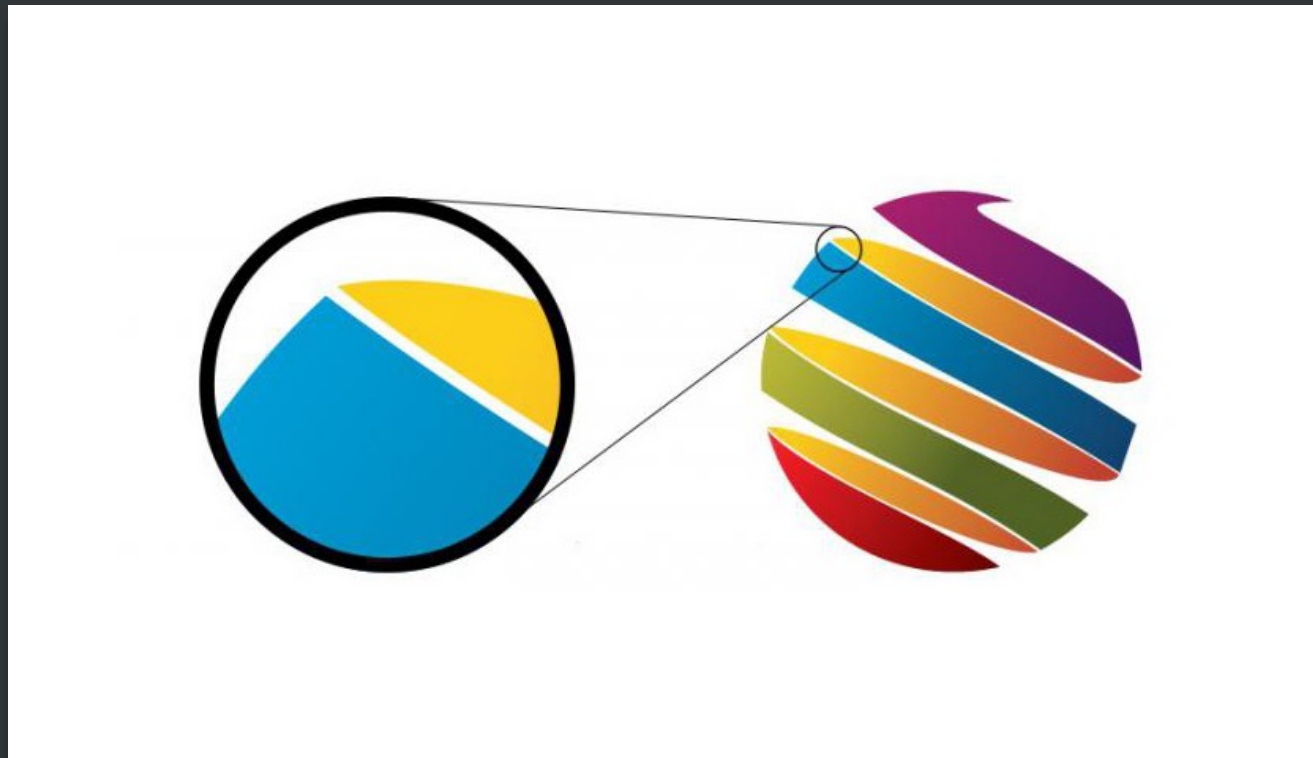
Тесселяция



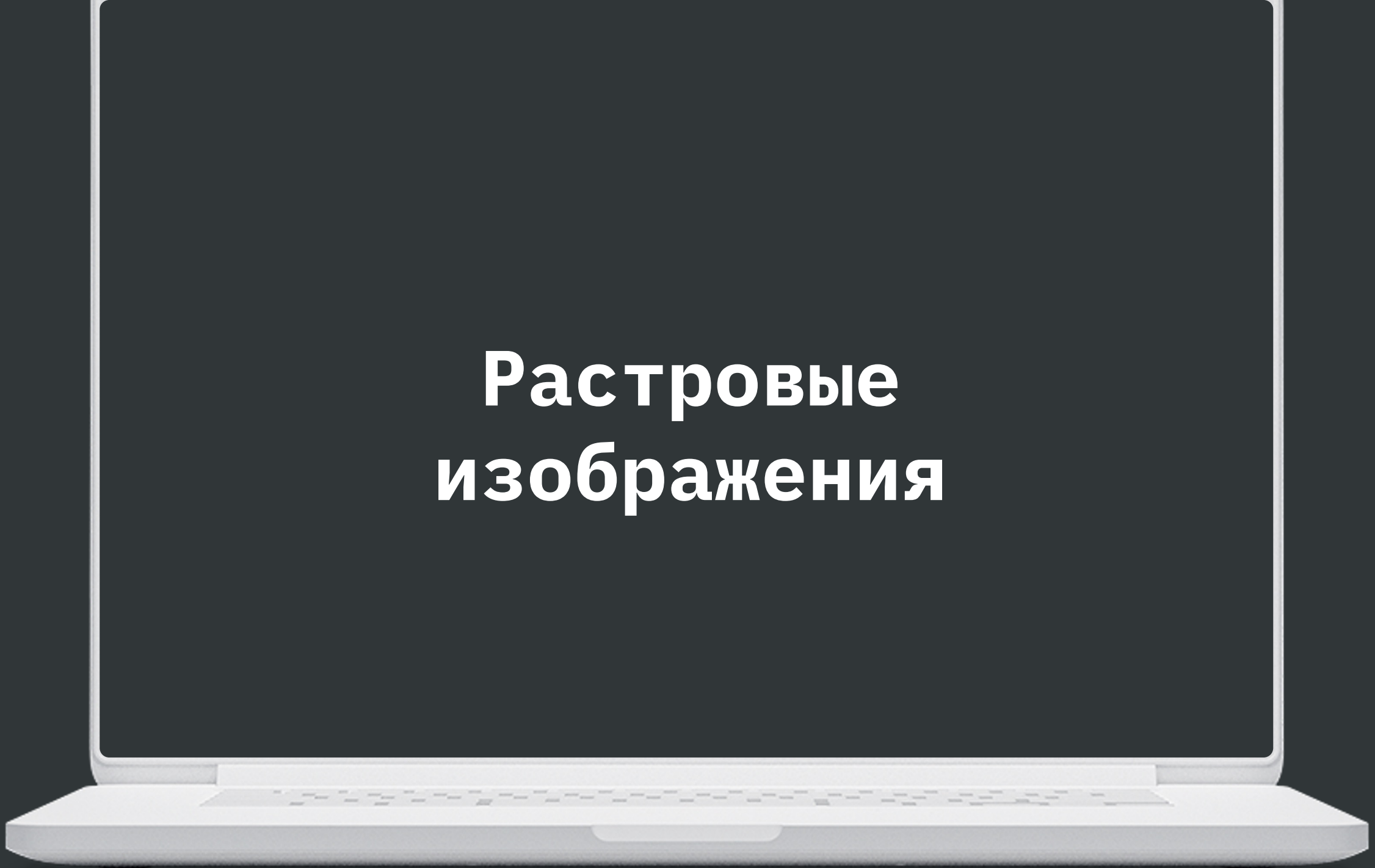
фрактальная графика



Векторная графика



Растровые изображения



Растровые изображения

Растровые изображения представляют собой массив (таблицу) пикселей разных цветов.



Растровая графика vs Векторной графики



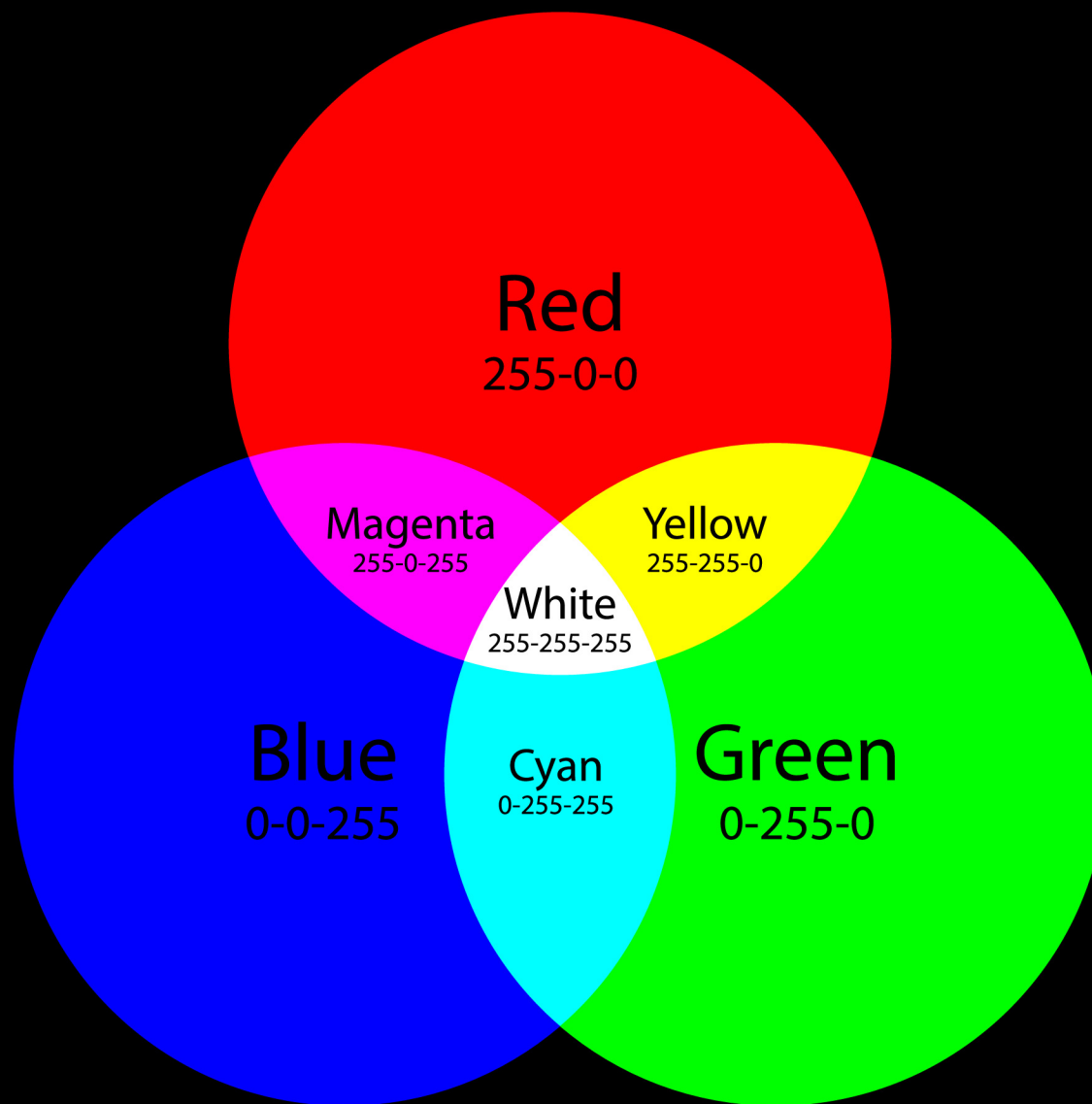
РАСТР
.jpeg .gif .png



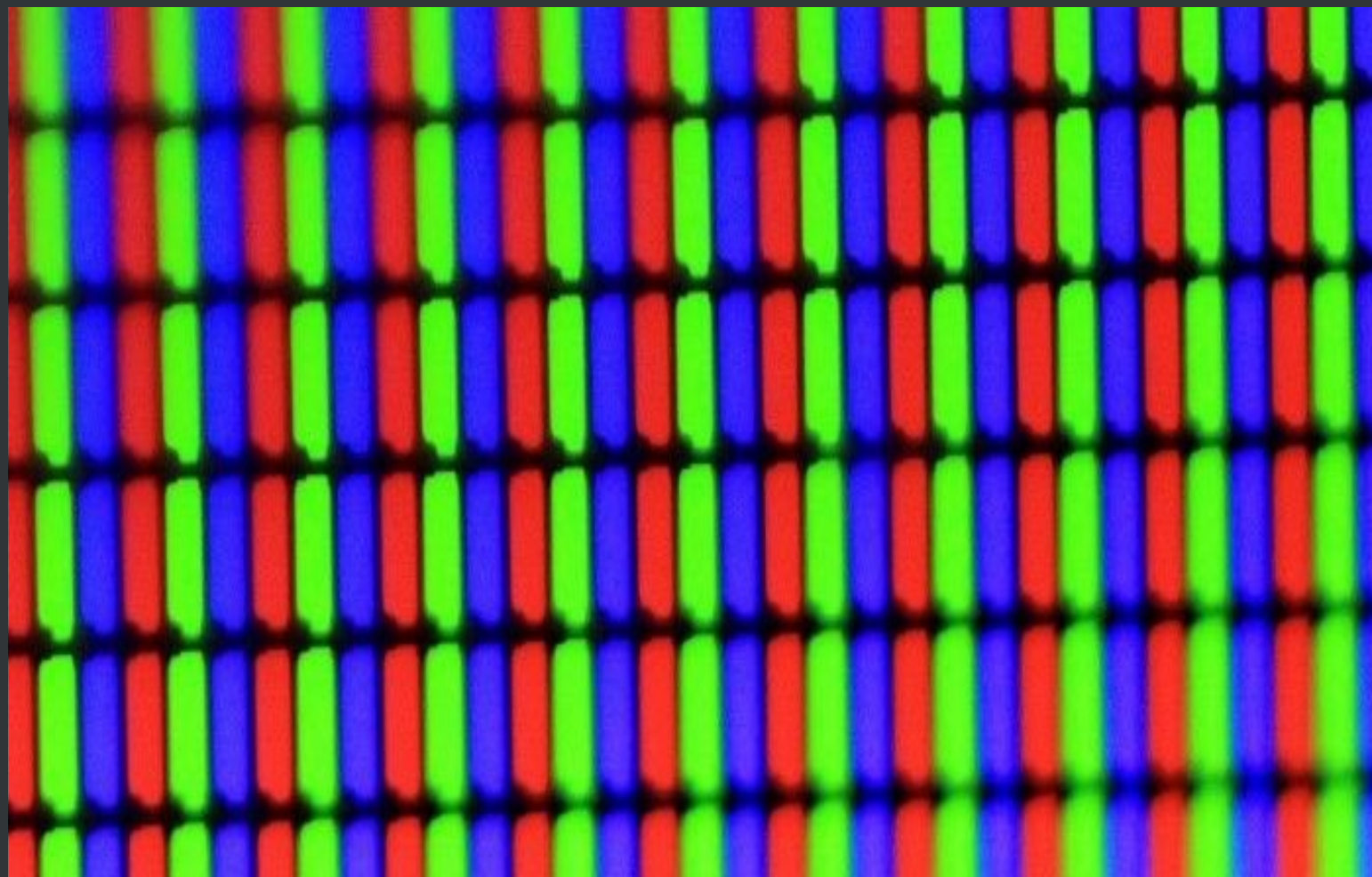
ВЕКТОР
.svg

R G B

Additive color mixing



R G B



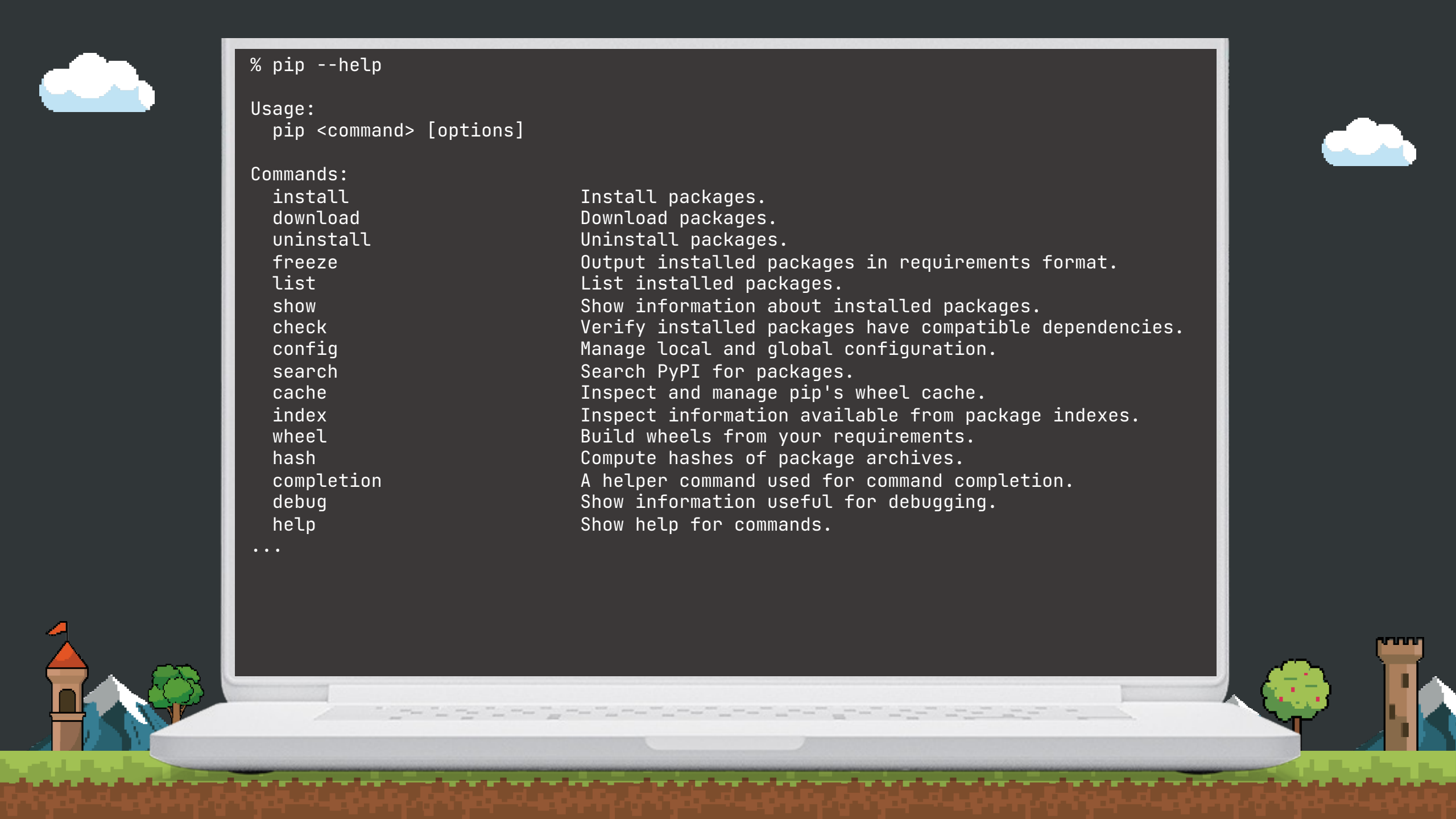
A white laptop is shown from a front-facing perspective, slightly angled. The screen is dark and displays the text 'PIL.' and 'Установка библиотек' in white. The keyboard and trackpad are visible on the laptop's base.

PIL.

Установка библиотек



```
% pip install pillow
```

```
% pip --help
```

```
Usage:
```

```
  pip <command> [options]
```

```
Commands:
```

```
  install
```

```
    Install packages.
```

```
  download
```

```
    Download packages.
```

```
  uninstall
```

```
    Uninstall packages.
```

```
  freeze
```

```
    Output installed packages in requirements format.
```

```
  list
```

```
    List installed packages.
```

```
  show
```

```
    Show information about installed packages.
```

```
  check
```

```
    Verify installed packages have compatible dependencies.
```

```
  config
```

```
    Manage local and global configuration.
```

```
  search
```

```
    Search PyPI for packages.
```

```
  cache
```

```
    Inspect and manage pip's wheel cache.
```

```
  index
```

```
    Inspect information available from package indexes.
```

```
  wheel
```

```
    Build wheels from your requirements.
```

```
  hash
```

```
    Compute hashes of package archives.
```

```
  completion
```

```
    A helper command used for command completion.
```

```
  debug
```

```
    Show information useful for debugging.
```

```
  help
```

```
    Show help for commands.
```

```
...
```



```
# Пример
```

```
from PIL import Image
```

```
im = Image.open('pic1.jpg')
```

```
pixels = im.load() # список с пикселями
```

```
x, y = im.size # ширина (x) и высота (y) изображения
```

```
for i in range(x):
```

```
    for j in range(y):
```

```
        r, g, b = pixels[i, j]
```

```
        pixels[i, j] = g, b, r
```

```
im.save('pic2.jpg')
```



```
# Создание изображений
```

```
from PIL import Image
```

```
im = Image.new("RGB", (500, 500), (0, 255, 0))
```

```
# Посмотрим, изображение какого размера у нас получилось
```

```
print(im.size)
```

```
im.save("res.jpg")
```

```
(500, 500)
```



```
# Кто быстрее?
```

```
from math import sqrt
```

```
# ex1
```

```
a = []
```

```
for i in range(1_000_000):  
    a.append(sqrt(i))
```

```
# ex2
```

```
[sqrt(x) for x in range(1_000_000)]
```

```
# ex3
```

```
list(map(sqrt, range(1_000_000)))
```





```
# Измерение скорости
```

```
from timeit import timeit
```

```
# ex1
```

```
print(timeit('for i in range(1_000_000): a.append(sqrt(i))',  
            'from math import sqrt; a = []', number=1))
```

```
# ex2
```

```
print(timeit('[sqrt(x) for x in range(1_000_000)]',  
            'from math import sqrt', number=1))
```

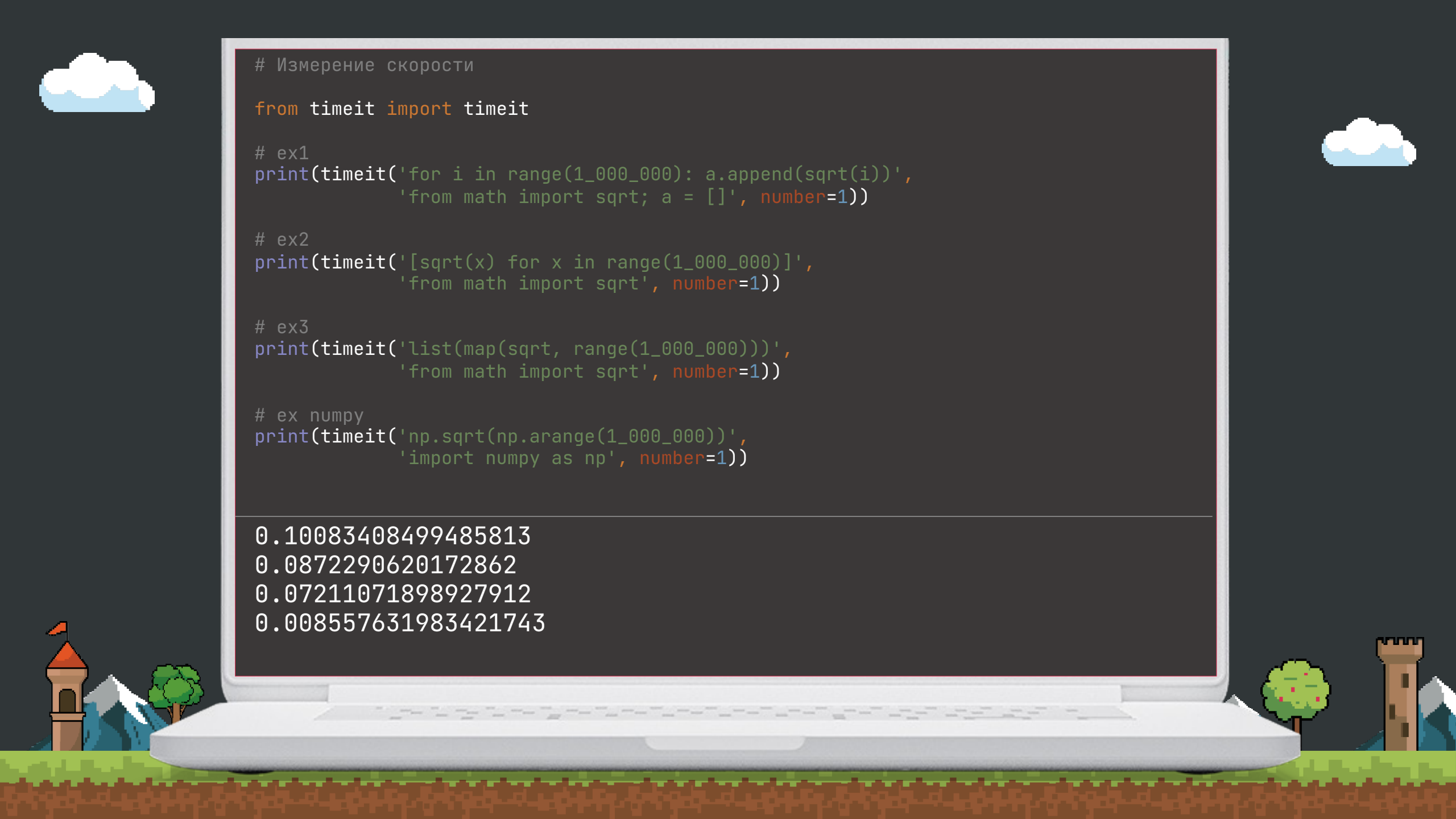
```
# ex3
```

```
print(timeit('list(map(sqrt, range(1_000_000)))',  
            'from math import sqrt', number=1))
```

```
0.10083408499485813
```

```
0.0872290620172862
```

```
0.07211071898927912
```



```
# Измерение скорости
```

```
from timeit import timeit
```

```
# ex1
```

```
print(timeit('for i in range(1_000_000): a.append(sqrt(i))',  
            'from math import sqrt; a = []', number=1))
```

```
# ex2
```

```
print(timeit('[sqrt(x) for x in range(1_000_000)]',  
            'from math import sqrt', number=1))
```

```
# ex3
```

```
print(timeit('list(map(sqrt, range(1_000_000)))',  
            'from math import sqrt', number=1))
```

```
# ex numpy
```

```
print(timeit('np.sqrt(np.arange(1_000_000))',  
            'import numpy as np', number=1))
```

```
0.10083408499485813
```

```
0.0872290620172862
```

```
0.07211071898927912
```

```
0.008557631983421743
```

TO BE CONTINUED...



Part 2

Предлагаю разобрать
задачи по анализу данных.

