ПРИЛОЖЕНИЕ

Ответы к упражнениям

Глава 2. Типы и переменные

2.1. Если вы еще не установили Python, сделайте это сейчас. Подробности, касающиеся установки, вы найдете в главе 1.

2.2. Запустите интерактивный интерпретатор Python. И вновь подробности вы найдете в главе 1. Интерпретатор должен вывести несколько строк о себе, а затем строку, начинающуюся с символов `>>>`. Это приглашение к вводу команд Python.

Вот так это выглядит на моем Mac:

```
$ python
Python 3.13.0 (v3.13.0:60403a5409f, Oct 7 2024, 00:37:40)
[Clang 15.0.0 (clang-1500.3.9.4)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

2.3. Немного поэкспериментируйте с интерпретатором. Используйте его как калькулятор и наберите текст `8 * 9`. Нажмите клавишу Enter, чтобы увидеть результат. Python должен вывести `72`.

```
>>> 8 * 9
72
```

2.4. Теперь введите число `47` и нажмите клавишу Enter. Появилось ли число `47` в следующей строке?

```
>>> 47
47
```

Теперь введите `print(47)` и нажмите клавишу Enter. Появилось ли снова число `47` в следующей строке?

```
>>> print(47)
47
```

Глава 3. Числа

3.1. Сколько секунд в часе? Используйте интерактивный интерпретатор как калькулятор и умножьте количество секунд в минуте (60) на количество минут в часе (тоже 60).

```
>>> 60 * 60
3600
```

3.2. Присвойте результат вычисления предыдущего задания (секунды в часе) переменной, которая называется `seconds_per_hour`.

```
>>> seconds_per_hour = 60 * 60
>>> seconds_per_hour
3600
```

3.3. Сколько секунд в сутках? Используйте переменную `seconds_per_hour`.

```
>>> seconds_per_hour * 24
86400
```

3.4. Снова посчитайте количество секунд в сутках, однако на этот раз сохраните результат в переменной `seconds_per_day`.

```
>>> seconds_per_day = seconds_per_hour * 24
>>> seconds_per_day
86400
```

3.5. Разделите значение переменной `seconds_per_day` на значение переменной `seconds_per_hour`. Используйте деление с плавающей точкой (`/`).

```
>>> seconds_per_day / seconds_per_hour
24.0
```

3.6. Разделите значение переменной `seconds_per_day` на значение переменной `seconds_per_hour`. Используйте целочисленное деление (`//`). Совпадает ли полученный результат с ответом на предыдущее упражнение, если не учитывать символы `.0` в конце?

```
>>> seconds_per_day // seconds_per_hour
24
```

Глава 4. Строки

4.1. Запишите с прописной буквы слово, которое начинается с буквы `m`:

```
>>> song = """When an eel grabs your arm,
... And it causes great harm,
... That's - a moray!"""
```

Не забудьте о пробеле, стоящем перед буквой `m`:

```
>>> song = """When an eel grabs your arm,
... And it causes great harm,
... That's - a moray!"""
>>> song = song.replace(" m", " M")
>>> print(song)
When an eel grabs your arm,
And it causes great harm,
That's - a Moray!
```

4.2. Выведите на экран следующее стихотворение, используя старый стиль форматирования. Подставьте в него строки `'roast beef'`, `'ham'`, `'head'` и `'clam'`:

```
My kitty cat likes %s,
My kitty cat likes %s,
My kitty cat fell on his %s
And now thinks he's a %s.
```

```
>>> poem = '''
... My kitty cat likes %s,
... My kitty cat likes %s,
... My kitty cat fell on his %s
... And now thinks he's a %s.
... '''
>>> args = ('roast beef', 'ham', 'head', 'clam')
>>> print(poem % args)

My kitty cat likes roast beef,
My kitty cat likes ham,
My kitty cat fell on his head
And now thinks he's a clam.
```

4.3. Напишите письмо, используя новый стиль форматирования. Сохраните следующую строку в переменной `letter` (она понадобится вам в упражнении, приведенном далее):

```
Dear {salutation} {name},

Thank you for your letter. We are sorry that our {product}
{verbed} in your {room}. Please note that it should never
be used in a {room}, especially near any {animals}.
```

```
Send us your receipt and {amount} for shipping and handling.
We will send you another {product} that, in our tests,
is {percent}% less likely to have {verbed}.

Thank you for your support.

Sincerely,
{spokesman}
{job_title}
```

```
>>> letter = '''
... Dear {salutation} {name},
...
... Thank you for your letter. We are sorry that our {product}
... {verbed} in your {room}. Please note that it should never
... be used in a {room}, especially near any {animals}.
...
... Send us your receipt and {amount} for shipping and handling.
... We will send you another {product} that, in our tests,
... is {percent}% less likely to have {verbed}.
...
... Thank you for your support.
...
... Sincerely,
... {spokesman}
... {job_title}
... '''
```

4.4. Присвойте значения переменным `salutation`, `name`, `product`, `verbed` (глагол в прошедшем времени), `room`, `animals`, `percent`, `spokesman` и `job_title`. Выведите на экран значение переменной `letter`, в которую подставлены эти значения, с помощью функции `letter.format()`.

```
>>> print (
...       letter.format(salutation='Ambassador',
...                             name='Nibbler',
...                             product='pudding',
...                             verbed='evaporated',
...                             room='gazebo',
...                             animals='octothorpes',
...                             amount='$1.99',
...                             percent=88,
...                             spokesman='Shirley Iugeste',
...                             job_title='I Hate This Job')
...       )
Dear Ambassador Nibbler,

Thank you for your letter. We are sorry that our pudding
evaporated in your gazebo. Please note that it should never
be used in a gazebo, especially near any octothorpes.
```

```
Send us your receipt and $1.99 for shipping and handling.
We will send you another pudding that, in our tests,
is 88% less likely to have evaporated.

Thank you for your support.

Sincerely,
Shirley Iugeste
I Hate This Job
```

4.5. Определите паттерн, который просматривается при подведении итогов голосования на лучшее имя для разных категорий: английская подводная лодка (Boaty McBoatface), австралийская беговая лошадь (Horsey McHorseface) и шведский поезд (Trainy McTrainface). Используйте форматирование с символом `%`, чтобы вывести на экран победившие имена для утки, тыквы и шпица, как показано в примере П.1.

Пример П.1. mcnames1.py

```
names = ["duck", "gourd", "spitz"]
for name in names:
      cap_name = name.capitalize()
      print("%sy Mc%sface" % (cap_name, cap_name))
```

Вывод:

```
Ducky McDuckface
Gourdy McGourdface
Spitzy McSpitzface
```

4.6. Сделайте то же самое с помощью функции `format()` (пример П.2).

Пример П.2. mcnames2.py

```
names = ["duck", "gourd", "spitz"]
for name in names:
      cap_name = name.capitalize()
      print("{}y Mc{}face".format(cap_name, cap_name))
```

4.7. А теперь еще раз с использованием f-строк (пример П.3).

Пример П.3. mcnames3.py

```
names = ["duck", "gourd", "spitz"]
for name in names:
      cap_name = name.capitalize()
      print(f"{cap_name}y Mc{cap_name}face")
```

Глава 5. Байтовые строки и массивы байтов

5.1. Создайте переменную типа `bytes` с именем `b` из целых чисел от 1 до 5.

```
>>> b = bytes((1,2,3,4,5))
>>> b
b'\x01\x02\x03\x04\x05'
>>> b = bytes(tuple(range(1,6)))
>>> b
b'\x01\x02\x03\x04\x05'
>>> b = bytes(list(range(1,6)))
>>> b
b'\x01\x02\x03\x04\x05'
```

5.2. Выведите длину `b`.

```
>>> len(b)
5
>>>
```

5.3. Создайте переменную типа `bytearray` с именем `ba`. Впрочем, вы можете дать ей любое другое имя, потому что это не имеет значения. Заполните переменную целыми числами от 1 до 5.

```
>>> ba = bytearray((1,2,3,4,5))
>>> ba
bytearray(b'\x01\x02\x03\x04\x05')
>>> ba = bytearray(tuple(range(1,6)))
>>> ba
bytearray(b'\x01\x02\x03\x04\x05')
>>> ba = bytearray(list(range(1,6)))
>>> ba
bytearray(b'\x01\x02\x03\x04\x05')
```

5.4. Выведите длину содержимого переменной `ba` (или как вы ее назвали).

```
>>> len(ba)
5
```

5.5. Присвойте переменной `ba` значение `b`.

```
>>> ba = bytearray(b)
>>> ba
bytearray(b'\x01\x02\x03\x04\x05')
```

5.6. Присвойте переменной b значение ba.

```
>>> b = bytes(ba)
>>> b
b'\x01\x02\x03\x04\x05'
```

5.7. Продублируйте содержимое b.

```
>>> b *= 2
>>> b
b'\x01\x02\x03\x04\x05\x01\x02\x03\x04\x05'
```

5.8. Продублируйте содержимое ba.

```
>>> ba = bytearray((1, 2, 3, 4, 5))
>>> ba *= 2
>>> ba
bytearray(b'\x01\x02\x03\x04\x05\x01\x02\x03\x04\x05')
```

Глава 6. Операторы if и match

6.1. Выберите число от 1 до 10 и присвойте его переменной secret. Затем выберите еще одно число от 1 до 10 и присвойте его переменной guess. Далее напишите условные проверки (if, else и elif), чтобы вывести строку 'too low', если значение переменной guess меньше значения secret, строку 'too high', если оно больше значения secret, и 'just right', если равно значению secret.

Выбрали ли вы для переменной secret значение 7? Готов поспорить, так сделали многие, поскольку в числе 7 есть нечто волшебное (пример П.4).

Пример П.4. guess.py

```
secret = 7
guess = 5
if guess < secret:
      print('too low')
elif guess > secret:
      print('too high')
else:
      print('just right')
```

Запустите эту программу — и вы увидите:

```
too low
```

6.2. Присвойте значения True или False переменным small и green. Напишите несколько операторов if/else, которые выведут на экран информацию о том,

соответствуют ли заданным условиям следующие растения: вишня, горох, арбуз, тыква.

```
>>> small = False
>>> green = True
>>> if small:
...       if green:
...             print("pea")
...       else:
...             print("cherry")
... else:
...       if green:
...             print("watermelon")
...       else:
...             print("pumpkin")
...
watermelon
```

Глава 7. Операторы while и for

7.1. Используйте цикл `for`, чтобы вывести на экран значения списка `[3, 2, 1, 0]`.

```
>>> for value in [3, 2, 1, 0]:
...       print(value)
...
3
2
1
0
```

7.2. Присвойте значение `7` переменной `guess_me` и значение `1` переменной `number`. Напишите цикл `while`, который сравнивает переменные `number` и `guess_me`. Выведите строку `'too low'`, если значение переменной `number` меньше значения `guess_me`. При равных значениях переменных `number` и `guess_me` выведите строку `'found it!'` и выйдите из цикла. Если значение переменной `number` больше значения `guess_me`, выведите строку `'oops'` и выйдите из цикла. Увеличьте значение переменной `number` в конце тела цикла (пример П.5).

Пример П.5. guess.py

```
guess_me = 7
number = 1
while True:
      if number < guess_me:
            print('too low')
      elif number == guess_me:
            print('found it!')
            break
```

```
    elif number > guess_me:
        print('oops')
        break
    number += 1
```

Если вы сделали все правильно, то увидите следующие строки:

```
too low
too low
too low
too low
too low
too low
found it!
```

Обратите внимание: строка `elif number > guess_me:` могла содержать обычный оператор `else:`, так как если значение `number` не меньше и не равно значению `guess_me`, то оно должно быть больше. По крайней мере в этой Вселенной.

7.3. Присвойте значение `5` переменной `guess_me`. Используйте цикл `for`, чтобы выполнить итерации с помощью переменной `number` по диапазону `range(10)`. Если значение переменной `number` меньше значения `guess_me`, то выведите на экран сообщение `'too low'`. Если оно равно значению `guess_me` — выведите сообщение `'found it!'`, а затем выйдите из цикла. Если значение переменной `number` больше, чем `guess_me`, то выведите на экран сообщение `'oops'` и выйдите из цикла.

```
>>> guess_me = 5
>>> for number in range(10):
...     if number < guess_me:
...         print("too low")
...     elif number == guess_me:
...         print("found it!")
...         break
...     else:
...         print("oops")
...         break
...
too low
too low
too low
too low
too low
found it!
```

Глава 8. Кортежи и списки

8.1. Создайте список, который называется `years_list`, содержащий год, в который вы родились, и каждый последующий год вплоть до вашего пятого дня рождения. Например, если вы родились в 1980 году, то список будет выглядеть так: `years_list = [1980, 1981, 1982, 1983, 1984, 1985]`.

Если вы родились в 1980-м, то вам нужно ввести следующее:

```
>>> years_list = [1980, 1981, 1982, 1983, 1984, 1985]
```

8.2. В какой из годов, содержащихся в списке `years_list`, был ваш третий день рождения? Помните, в первый год вам было 0 лет.

Вам нужно смещение 3. Поэтому, если вы родились в 1980-м:

```
>>> years_list[3]
1983
```

8.3. В какой из годов, содержащихся в списке `years_list`, вам было больше всего лет?

Вам нужно получить последний год, поэтому используйте смещение –1. Вы также можете взять смещение 5, поскольку знаете, что в этом списке всего шесть элементов. Однако смещение –1 позволяет получить последний элемент из списка любой длины. Для тех, кто родился в 1980 году:

```
>>> years_list[-1]
1985
```

8.4. Создайте и выведите список `things`, содержащий три элемента: `"mozzarella"`, `"cinderella"`, `"salmonella"`.

```
>>> things = ["mozzarella", "cinderella", "salmonella"]
>>> things
['mozzarella', 'cinderella', 'salmonella']
```

8.5. Напишите с большой буквы тот элемент списка `things`, который относится к человеку, а затем выведите список. Изменился ли элемент списка?

Эта строка записывает слово с прописной буквы, но не меняет его в списке:

```
>>> things[1].capitalize()
'Cinderella'
>>> things
['mozzarella', 'cinderella', 'salmonella']
```

Если хотите изменить его в списке, вам нужно присвоить его снова:

```
>>> things[1] = things[1].capitalize()
>>> things
['mozzarella', 'Cinderella', 'salmonella']
```

8.6. Переведите сырный элемент списка `things` в верхний регистр целиком и выведите список.

```
>>> things[0] = things[0].upper()
>>> things
['MOZZARELLA', 'Cinderella', 'salmonella']
```

8.7. Удалите возбудитель болезни из списка `things`, получите Нобелевскую премию и затем выведите список на экран.

Этот вызов удалит элемент по значению:

```
>>> things.remove("salmonella")
>>> things
['MOZZARELLA', 'Cinderella']
```

Поскольку элемент находится на последнем месте в списке, можно использовать такой прием:

```
>>> del things[-1]
```

Элемент также можно удалить, указав смещение от начала:

```
>>> del things[2]
```

8.8. Создайте список, который называется `surprise` и содержит элементы `"Groucho"`, `"Chico"` и `"Harpo"`.

```
>>> surprise = ['Groucho', 'Chico', 'Harpo']
>>> surprise
['Groucho', 'Chico', 'Harpo']
```

8.9. Преобразуйте первую букву последнего элемента списка `surprise` в нижний регистр, затем выведите эту строку в обратном порядке и с прописной буквы.

```
>>> surprise[-1] = surprise[-1].lower()
>>> surprise[-1] = surprise[-1][::-1]
>>> surprise[-1].capitalize()
'Oprah'
```

8.10. Используя списковое включение, создайте список `even`, в котором будут содержаться четные числа в диапазоне `range(10)`.

```
>>> even = [number for number in range(10) if number % 2 == 0]
>>> even
[0, 2, 4, 6, 8]
```

8.11. Попробуйте создать генератор рифм для прыжков через скакалку. Напечатайте последовательность двухстрочных рифм (пример П.6). Начните программу с этого фрагмента:

```
start1 = ["fee", "fie", "foe"]
rhymes = [
    ("flop", "get a mop"),
    ("fope", "turn the rope"),
    ("fa", "get your ma"),
    ("fudge", "call the judge"),
    ("fat", "pet the cat"),
    ("fog", "walk the dog"),
    ("fun", "say we're done"),
    ]
start2 = "Someone better"
```

Затем следуйте инструкциям.

Для каждого кортежа (`first, second`) в списке `rhymes`:

- для первой строки:
  - выведите на экран каждую строку из списка `start1` с прописной буквы и закончите восклицательным знаком и пробелом;
  - выведите на экран значение переменной `first`, также записав его с прописной буквы и с восклицательным знаком в конце;
- для второй строки:
  - выведите на экран значение переменной `start2` и пробел;
  - выведите на экран значение переменной `second` и точку.

Пример П.6. jump.py

```
start1 = ["fee", "fie", "foe"]
rhymes = [
      ("flop", "get a mop"),
       ("fope", "turn the rope"),
       ("fa", "get your ma"),
       ("fudge", "call the judge"),
       ("fat", "pet the cat"),
       ("fog", "walk the dog"),
       ("fun", "say we're done"),
      ]
start2 = "Someone better"
start1_caps = " ".join([word.capitalize() + "!" for word in start1])
for first, second in rhymes:
      print(f"{start1_caps} {first.capitalize()}!")
      print(f"{start2} {second}.")
```

Результат:

```
Fee! Fie! Foe! Flop!
Someone better get a mop.
Fee! Fie! Foe! Fope!
Someone better turn the rope.
Fee! Fie! Foe! Fa!
Someone better get your ma.
Fee! Fie! Foe! Fudge!
Someone better call the judge.
Fee! Fie! Foe! Fat!
Someone better pet the cat.
Fee! Fie! Foe! Fog!
Someone better walk the dog.
Fee! Fie! Foe! Fun!
Someone better say we're done.
```

8.12. Выведите каждый вопрос из списка с соответствующим ему ответом в следующей форме:

Q: *вопрос*

A: *ответ*

```
>>> questions = [
...      "We don't serve strings around here. Are you a string?",
...      "What is said on Father's Day in the forest?",
...      "What makes the sound 'Sis! Boom! Bah!'?"
...      ]
>>> answers = [
...      "An exploding sheep.",
...      "No, I'm a frayed knot.",
...      "'Pop!' goes the weasel."
...      ]
```

Вывести каждый элемент из `questions` вместе с его парой из `answers` можно разными способами. Попробуем использовать сэндвич из кортежей (кортежи в кортеже) для их объединения в пары и распаковку кортежей для извлечения и вывода (пример П.7).

Пример П.7. qanda.py

```
questions = [
      "We don't serve strings around here. Are you a string?",
      "What is said on Father's Day in the forest?",
      "What makes the sound 'Sis! Boom! Bah!'?"
      ]
answers = [
      "An exploding sheep.",
```

```
      "No, I'm a frayed knot.",
      "'Pop!' goes the weasel."
      ]

q_a = ( (0, 1), (1,2), (2, 0) )
for q, a in q_a:
      print("Q:", questions[q])
      print("A:", answers[a])
      print()
```

Вывод:

```
$ python qanda.py
Q: We don't serve strings around here. Are you a string?
A: No, I'm a frayed knot.

Q: What is said on Father's Day in the forest?
A: 'Pop!' goes the weasel.

Q: What makes the sound 'Sis! Boom! Bah!'?
A: An exploding sheep.
```

Глава 9. Словари и множества

9.1. Создайте англо-французский словарь `e2f` и выведите его на экран. Вот ваши первые слова: `dog/chien`, `cat/chat` и `walrus/morse`.

```
>>> e2f = {'dog': 'chien', 'cat': 'chat', 'walrus': 'morse'}
>>> e2f
{'cat': 'chat', 'walrus': 'morse', 'dog': 'chien'}
```

9.2. Используя словарь `e2f`, выведите французский вариант слова `walrus`.

```
>>> e2f['walrus']
'morse'
```

9.3. Создайте франко-английский словарь `f2e` на основе словаря `e2f`. Задействуйте метод `items`.

```
>>> f2e = {}
>>> for english, french in e2f.items():
... f2e[french] = english
>>> f2e
{'morse': 'walrus', 'chien': 'dog', 'chat': 'cat'}
```

9.4. Используя словарь `f2e`, выведите английский вариант слова `chien`.

```
>>> f2e['chien']
'dog'
```

9.5. Создайте и выведите на экран множество английских слов из ключей словаря `e2f`.

```
>>> set(e2f.keys())
{'cat', 'walrus', 'dog'}
```

9.6. Создайте многоуровневый словарь `life`. Используйте следующие строки для ключей верхнего уровня `'animals'`, `'plants'` и `'other'`. Сделайте так, чтобы ключ `'animals'` ссылался на другой словарь, имеющий ключи `'cats'`, `'octopi'` и `'emus'`. Сделайте так, чтобы ключ `'cats'` ссылался на список строк со значениями `'Henri'`, `'Grumpy'` и `'Lucy'`. Остальные ключи должны ссылаться на пустые словари.

Это довольно сложный пример, так что если вы заглянули сюда, то ничего особо страшного не случилось:

```
>>> life = {
...       'animals': {
...             'cats': [
...                   'Henri', 'Grumpy', 'Lucy'
...                   ],
...             'octopi': {},
...             'emus': {}
...             },
...       'plants': {},
...       'other': {}
...       }
>>>
```

9.7. Выведите на экран высокоуровневые ключи словаря `life`.

```
>>> print(life.keys())
dict_keys(['animals', 'other', 'plants'])
```

Python 3 имеет все необходимое для работы с ключами словарей. Вот как вывести их в виде простого списка:

```
>>> print(list(life.keys()))
['animals', 'other', 'plants']
```

Вы можете использовать пробелы, чтобы сделать код более читабельным:

```
>>> print ( list ( life.keys() ) )
['animals', 'other', 'plants']
```

9.8. Выведите на экран ключи `life['animals']`.

```
>>> print(life['animals'].keys())
dict_keys(['cats', 'octopi', 'emus'])
```

9.9. Выведите значения `life['animals']['cats']`.

```
>>> print(life['animals']['cats'])
['Henri', 'Grumpy', 'Lucy']
```

9.10. Используйте генератор словаря, чтобы создать словарь `squares`. Задействуйте вызов `range(10)`, чтобы получить ключи. В качестве значений возьмите возведенное в квадрат значение каждого ключа.

```
>>> squares = {key: key*key for key in range(10)}
>>> squares
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
```

9.11. Используйте генератор множества, чтобы создать множество `odd` из нечетных чисел, входящих в диапазон `range(10)`.

```
>>> odd = {number for number in range(10) if number % 2 == 1}
>>> odd
{1, 3, 5, 7, 9}
```

9.12. Используйте включение генератора, чтобы вернуть строку `'Got'` и числа из диапазона `range(10)`. Выполните обход содержимого этой конструкции с помощью цикла `for`.

```
>>> for thing in ('Got %s' % number for number in range(10)):
...       print(thing)
...
Got 0
Got 1
Got 2
Got 3
Got 4
Got 5
Got 6
Got 7
Got 8
Got 9
```

9.13. Используйте функцию `zip()`, чтобы создать словарь из кортежа ключей `('optimist', 'pessimist', 'troll')` и кортежа значений `('The glass is half full', 'The glass is half empty', 'How did you get a glass?')`.

```
>>> keys = ('optimist', 'pessimist', 'troll')
>>> values = ('The glass is half full',
...       'The glass is half empty',
...       'How did you get a glass?')
>>> dict(zip(keys, values))
{'optimist': 'The glass is half full',
'pessimist': 'The glass is half empty',
'troll': 'How did you get a glass?'}
```

9.14. Используйте функцию `zip()`, чтобы создать словарь `movies`, в котором объединены списки `titles = ['Creature of Habit', 'Crewel Fate', 'Sharks On a Plane']` и `plots = ['A nun turns into a monster', 'A haunted yarn shop', 'Check your exits']`.

```
>>> titles = ['Creature of Habit',
...       'Crewel Fate',
...       'Sharks On a Plane']
>>> plots = ['A nun turns into a monster',
...       'A haunted yarn shop',
...       'Check your exits']
>>> movies = dict(zip(titles, plots))
>>> movies
{'Creature of Habit': 'A nun turns into a monster',
'Crewel Fate': 'A haunted yarn shop',
'Sharks On a Plane': 'Check your exits'}
>>>
```

Глава 10. Функции

10.1. Определите функцию `good`, которая возвращает список `['Harry', 'Ron', 'Hermione']`.

```
>>> def good():
...       return ['Harry', 'Ron', 'Hermione']
...
>>> good()
['Harry', 'Ron', 'Hermione']
```

10.2. Определите функцию генератора `get_odds`, которая возвращает нечетные числа из диапазона `range(10)`. Используйте цикл `for`, чтобы найти и вывести третье возвращенное значение.

```
>>> def get_odds():
...       for number in range(1, 10, 2):
...             yield number
...
>>> count = 1
>>> for number in get_odds():
...       if count == 3:
...             print("The third odd number is", number)
...             break
...       count += 1
...
The third odd number is 5.
```

10.3. Определите декоратор `test`, который выводит строку `'start'`, когда вызывается функция, и строку `'end'`, когда функция завершает свою работу.

```
>>> def test(func):
...       def new_func(*args, **kwargs):
...             print('start')
...             result = func(*args, **kwargs)
...             print('end')
...             return result
...       return new_func
...
>>>
>>> @test
... def greeting():
...       print("Greetings, Earthling")
...
>>> greeting()
start
Greetings, Earthling
end
```

10.4. Определите исключение `OopsException`. Сгенерируйте его и посмотрите, что произойдет. Затем напишите код, позволяющий поймать это исключение и вывести строку `'Caught an oops'`.

```
>>> class OopsException(Exception):
...       pass
...
>>> raise OopsException()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
__main__.OopsException
>>>
>>> try:
...       raise OopsException
... except OopsException:
...       print('Caught an oops')
...
Caught an oops
```

Глава 11. Объекты

11.1. Создайте класс `Thing`, не имеющий содержимого, и выведите его на экран. Затем создайте объект `example` этого класса и также выведите его. Совпадают ли выведенные значения?

```
>>> class Thing:
...       pass
...
>>> print(Thing)
<class '__main__.Thing'>
>>> example = Thing()
>>> print(example)
<__main__.Thing object at 0x1006f3fd0>
```

11.2. Создайте новый класс `Thing2` и присвойте атрибуту `letters` значение `'abc'`. Выведите на экран значение атрибута `letters`.

```
>>> class Thing2:
...       letters = 'abc'
...
>>> print(Thing2.letters)
abc
```

11.3. Создайте еще один класс, конечно же `Thing3`. В этот раз присвойте значение `'xyz'` атрибуту экземпляра (объекта) `letters`. Выведите на экран значение атрибута `letters`.

Понадобилось ли вам создавать объект класса, чтобы сделать это?

```
>>> class Thing3:
...       def __init__(self):
...             self.letters = 'xyz'
...
```

Переменная `letters` принадлежит объекту класса `Thing3`, но не самому классу `Thing3`:

```
>>> print(Thing3.letters)
Traceback (most recent call last):
   File "<stdin>", line 1, in <module>
AttributeError: type object 'Thing3' has no attribute 'letters'
>>> something = Thing3()
>>> print(something.letters)
xyz
```

11.4. Создайте класс `Element`, имеющий атрибуты экземпляра `name`, `symbol` и `number`. Создайте объект этого класса со значениями `'Hydrogen'`, `'H'` и `1`.

```
>>> class Element:
...       def __init__(self, name, symbol, number):
...             self.name = name
...             self.symbol = symbol
...             self.number = number
...
>>> hydrogen = Element('Hydrogen', 'H', 1)
```

11.5. Создайте словарь со следующими ключами и значениями: `'name': 'Hydrogen'`, `'symbol': 'H'`, `'number': 1`. Затем создайте объект класса `Element` с именем `hydrogen`, использовав этот словарь.

Начнем со словаря:

```
>>> el_dict = {'name': 'Hydrogen', 'symbol': 'H', 'number': 1}
```

Задачу можно решить так, однако придется ввести много текста:

```
>>> hydrogen = Element(el_dict['name'], el_dict['symbol'], el_dict['number'])
```

Убедимся, что все получилось:

```
>>> hydrogen.name
'Hydrogen'
```

Вы также можете инициализировать объект непосредственно с помощью словаря, поскольку имена его ключей совпадают с именами аргументов функции `__init__` (именованные аргументы рассматриваются в главе 10):

```
>>> hydrogen = Element(**el_dict)
>>> hydrogen.name
'Hydrogen'
```

11.6. Определите в классе `Element` метод `dump()`, который выводит на экран значения атрибутов экземпляра (`name`, `symbol` и `number`). Создайте объект `hydrogen` этого обновленного класса и используйте метод `dump()`, чтобы вывести на экран его атрибуты.

```
>>> class Element:
...       def __init__(self, name, symbol, number):
...             self.name = name
...             self.symbol = symbol
...             self.number = number
...       def dump(self):
...             print('name=%s, symbol=%s, number=%s' %
...                       (self.name, self.symbol, self.number))
...
>>> hydrogen = Element(**el_dict)
>>> hydrogen.dump()
name=Hydrogen, symbol=H, number=1
```

11.7. Вызовите функцию `print(hydrogen)`. В определении класса `Element` измените имя метода `dump` на `__str__`, создайте новый объект `hydrogen` и затем снова вызовите метод `print(hydrogen)`.

```
>>> print(hydrogen)
<__main__.Element object at 0x1006f5310>
>>> class Element:
...       def __init__(self, name, symbol, number):
...             self.name = name
...             self.symbol = symbol
...             self.number = number
...       def __str__(self):
...             return ('name=%s, symbol=%s, number=%s' %
...                       (self.name, self.symbol, self.number))
...
>>> hydrogen = Element(**el_dict)
>>> print(hydrogen)
name=Hydrogen, symbol=H, number=1
```

Метод `__str__()` — это один из *волшебных методов* Python. Функция `print` вызывает метод объекта `__str__()`, чтобы получить его строковое представление. Если у объекта нет метода `__str__()`, то он наследует метод от родительского класса `Object`, который возвращает строку наподобие `<__main__.Element object at 0x1006f5310>`.

11.8. Модифицируйте класс `Element`, сделав атрибуты `name`, `symbol` и `number` закрытыми. Определите для каждого атрибута метод-геттер, возвращающий его значение.

```
>>> class Element:
...       def __init__(self, name, symbol, number):
...             self.__name = name
...             self.__symbol = symbol
...             self.__number = number
...       @property
...       def name(self):
...             return self.__name
...       @property
...       def symbol(self):
...             return self.__symbol
...       @property
...       def number(self):
...             return self.__number
...
>>> hydrogen = Element('Hydrogen', 'H', 1)
>>> hydrogen.name
'Hydrogen'
>>> hydrogen.symbol
'H'
>>> hydrogen.number
1
```

11.9. Определите три класса: `Bear`, `Rabbit` и `Octothorpe`. В каждом определите всего один метод — `eats()`. Этот метод должен возвращать значения `'berries'` (для `Bear`), `'clover'` (для `Rabbit`) или `'campers'` (для `Octothorpe`). Создайте по одному объекту каждого класса и выведите на экран то, что ест каждое животное.

```
>>> class Bear:
...       def eats(self):
...             return 'berries'
...
>>> class Rabbit:
...       def eats(self):
...             return 'clover'
...
>>> class Octothorpe:
...       def eats(self):
...             return 'campers'
...
>>> b = Bear()
>>> r = Rabbit()
>>> o = Octothorpe()
>>> print(b.eats())
berries
>>> print(r.eats())
clover
>>> print(o.eats())
campers
```

11.10. Определите три класса: `Laser`, `Claw` и `SmartPhone`. В каждом определите всего один метод — `does()`. Он должен возвращать значения `'disintegrate'` (для `Laser`), `'crush'` (для `Claw`) или `'ring'` (для `SmartPhone`). Далее определите класс `Robot`, содержащий по одному экземпляру (объекту) каждого из этих классов. Определите метод `does()` в классе `Robot`, который выводит на экран все, что делают его компоненты.

```
>>> class Laser:
...       def does(self):
...             return 'disintegrate'
...
>>> class Claw:
...       def does(self):
...             return 'crush'
...
>>> class SmartPhone:
...       def does(self):
...             return 'ring'
...
>>> class Robot:
...       def __init__(self):
...             self.laser = Laser()
...             self.claw = Claw()
...             self.smartphone = SmartPhone()
```

```
...       def does(self):
...             return '''I have many attachments:
... My laser, to %s.
... My claw, to %s.
... My smartphone, to %s.''' % (
...       self.laser.does(),
...       self.claw.does(),
...       self.smartphone.does() )
...
>>> robbie = Robot()
>>> print( robbie.does() )
I have many attachments:
My laser, to disintegrate.
My claw, to crush.
My smartphone, to ring.
```

Глава 12. Модули и пакеты

12.1. Создайте файл `zoo.py`. В нем объявите функцию `hours()`, которая выводит на экран строку `'Open 9-5 daily'`. Далее используйте интерактивный интерпретатор, чтобы импортировать модуль `zoo` и вызвать его функцию `hours()`.

Содержимое `zoo.py` показано в примере П.8.

Пример П.8. zoo.py

```
def hours():
      print('Open 9-5 daily')
```

А теперь импортируем его интерактивно:

```
>>> import zoo
>>> zoo.hours()
Open 9-5 daily
```

12.2. В интерактивном интерпретаторе импортируйте модуль `zoo` под именем `menagerie` и вызовите его функцию `hours()`.

```
>>> import zoo as menagerie
>>> menagerie.hours()
Open 9-5 daily
```

12.3. Оставаясь в интерпретаторе, импортируйте непосредственно функцию `hours()` из модуля `zoo` и вызовите ее.

```
>>> from zoo import hours
>>> hours()
Open 9-5 daily
```

12.4. Импортируйте функцию `hours()` под именем `info` и вызовите ее.

```
>>> from zoo import hours as info
>>> info()
Open 9-5 daily
```

12.5. Создайте словарь `plain`, содержащий пары «ключ — значение» `'a': 1, 'b': 2` и `'c':3`, а затем выведите его на экран.

```
>>> plain = {'a': 1, 'b': 2, 'c': 3}
>>> plain
{'a': 1, 'c': 3, 'b': 2}
```

12.6. Создайте `OrderedDict` с именем `fancy` из пар «ключ — значение», приведенных в предыдущем упражнении, и выведите его на экран. Изменился ли порядок ключей?

```
>>> from collections import OrderedDict
>>> fancy = OrderedDict([('a', 1), ('b', 2), ('c', 3)])
>>> fancy
OrderedDict([('a', 1), ('b', 2), ('c', 3)])
```

12.7. Создайте `defaultdict` с именем `dict_of_lists` и передайте ему аргумент `list`. Создайте список `dict_of_lists['a']` и присоедините к нему значение `'something for a'` одним присваиванием. Выведите на экран `dict_of_lists['a']`.

```
>>> from collections import defaultdict
>>> dict_of_lists = defaultdict(list)
>>> dict_of_lists['a'].append('something for a')
>>> dict_of_lists['a']
['something for a']
```

Глава 13. Среда разработки

13.1. Установите `pip`, если у вас его нет. С помощью `pip` установите пакет `pyspellchecker`.

```
$ pip install pyspellchecker
```

13.2. Ознакомьтесь с документацией по `pyspellchecker` на сайтах PyPI (https://oreil.ly/YjhXb) и Read the Docs (https://oreil.ly/AXtNt). Напишите небольшую программу, которая получает список слов, сообщает, есть ли в них ошибки,

и выводит возможное правильное написание. Обратите внимание на то, что пакет `pyspellchecker` поддерживает более десяти естественных языков (пример П.9).

Пример П.9. check.py

```
import spellchecker as sp

ch = sp.SpellChecker()
words = ('mispell', 'phantom', 'tumeric', 'dilatation')
wrong = ch.unknown(words)
for word in words:
      print(word)
      if word in wrong:
            print(' '*4, ch.correction(word), ch.candidates(word))
```

```
$ python check.py
mispell
      misspell {misspell, ispell}
phantom
tumeric
      numeric {numeric, turmeric}
dilatation
```

13.3. (Не обязательно.) Установите менеджер пакетов uv (https://oreil.ly/iZkJ8). Используйте его для проверки корректности программы, написанной в упражнении 13.2.

13.4. (Не обязательно.) Установите PyCharm (https://oreil.ly/13uQ1) и VS Code (https://oreil.ly/XP0r_). Попробуйте использовать их.

Глава 14. Аннотации типов и документация

14.1. Напишите функцию `pointless()`, которая принимает словарь элементов типа `str:int`, преобразует первый символ ключа в верхний регистр, прибавляет 1 к значению и выводит ключ и значение. Добавьте полноценные аннотации типов (пример П.10).

Пример П.10. hints.py

```
def pointless(data: dict[str, int]) -> None:
      for key, val in data.items():
            print(key.capitalize(), val + 1)

data = {'hawaii':49, 'carbon': 13}
pointless(data)
```

```
$ python hints.py
Hawaii 50
Carbon 14
```

Глава 15. Тестирование

15.1. Найдите ошибку в примере П.11. Используйте Pylint, Pyflakes и Ruff (установите их при необходимости).

Пример П.11. errors.py

```
import math

print("square root of 3 is", sqrt(3))
```

```
$ pylint errors.py
* Module errors
errors.py:1:0: C0114: Missing module docstring (missing-module-docstring)
errors.py:3:29: E0602: Undefined variable sqrt (undefined-variable)
errors.py:1:0: W0611: Unused import math (unused-import)

-----------------------------------
Your code has been rated at 0.00/10
```

```
$ pyflakes errors.py
errors.py:1:1: math imported but unused
errors.py:3:30: undefined name sqrt
```

```
$ ruff check errors.py
errors.py:1:8: F401 [*] `math` imported but unused
  |
1 | import math
  |        ^^ F401
2 |
3 | print("square root of 3 is", sqrt(3))
  |
  = help: Remove unused import: `math`

errors.py:3:30: F821 Undefined name `sqrt`
  |
1 | import math
2 |
3 | print("square root of 3 is", sqrt(3))
  |                              ^^ F821
  |

Found 2 errors.
[*] 1 fixable with the `--fix` option.
```

15.2. Создайте файл с кодом на Python (пример П.12), содержащий:

- функцию, которая складывает два целых числа и возвращает их сумму;
- несколько тестов pytest для этой функции;
- фикстуру, которая возвращает текущую дату и время вызова функции.

Пример П.12. test_arino.py

```
import pytest
import datetime

def add_integers(a, b):
    return a + b

@pytest.fixture(scope="module")
def curtime():
    return datetime.datetime.now(datetime.UTC)

def test_zeroes(curtime):
    print(f"{curtime=}")
    assert(add_integers(0, 0) == 0)

def test_positive_ints(curtime):
    print(f"{curtime=}")
    assert(add_integers(5, 6) == 11)

def test_negative_ints(curtime):
    print(f"{curtime=}")
    assert(add_integers(-1, -1) == -2)
```

```
$ pytest -sv test_arino.py
============================ test session starts ============================
...
collected 3 items

test_arino.py::test_zeroes curtime = 2025-06-01 18:46:04.525185+00:00
PASSED
test_arino.py::test_positive_ints curtime = 2025-06-01 18:46:04.525185+00:00
PASSED
test_arino.py::test_negative_ints curtime = 2025-06-01 18:46:04.525185+00:00
PASSED
============================ 3 passed in 0.20s ============================
```

Глава 16. Отладка

16.1. Используйте f-строки для вывода следующих выражений:

- присвойте значение 5 переменной x, затем выведите значение в два раза большее, если x нечетное, в противном случае выведите 2;
- выведите значение x в поле шириной 20 пробелов, выровняв по центру и заменив точками пробелы слева и справа.

```
>>> x = 5
>>> print(f"{2*x if x%2 == 1 else 2}")
10
>>> print(f"{x:.^20}")
.........5.........
```

Глава 17. Текстовые данные

17.1. Создайте строку Юникода с именем `mystery` и присвойте ей значение `\U0001f984`. Выведите на экран значение строки `mystery` и имя символа Юникода в ней. Затем присвойте переменной `mystery2` значение `\U0001f4a9` и сделайте с ней то же самое. (Имейте в виду, что в зависимости от используемого системного шрифта ваш код может вывести или не вывести один или оба символа.)

```
>>> import unicodedata
>>> mystery = '\U0001f984'
>>> print(mystery)
🦄
>>> unicodedata.name(mystery)
'UNICORN FACE'
>>> mystery = '\U0001f4a9'
>>> mystery
'💩'
>>> unicodedata.name(mystery)
'PILE OF POO'
```

Черт побери! Чего только там нет!

17.2. Закодируйте строку `mystery`, в этот раз с использованием кодировки UTF-8, в переменную типа `bytes` с именем `pop_bytes`. Выведите значение `pop_bytes`.

```
>>> pop_bytes = mystery.encode('utf-8')
>>> pop_bytes
b'\xf0\x9f\x92\xa9'
```

17.3. Используя кодировку UTF-8, декодируйте переменную `pop_bytes` в строку `pop_string`. Выведите значение `pop_string`. Сравните ее значение со значением переменной `mystery`.

```
>>> pop_string = pop_bytes.decode('utf-8')
>>> pop_string
🦄
>>> pop_string == mystery
True
```

17.4. Во время работы с текстом часто бывает удобно использовать регулярные выражения. Применим их к примеру текста — стихотворению *Ode on the Mammoth Cheese*, написанному Джеймсом Макинтайром в 1866 году во славу головки сыра весом 3180 килограммов, изготовленной в Онтарио и отправленной в международное путешествие. Если не хотите вводить это стихотворение целиком, то используйте свой любимый поисковик и скопируйте текст в программу. Или просто

скачайте текст на сайте Project Gutenberg (https://bit.ly/mcintyre-poetry). Присвойте текст переменной с именем `mammoth`.

```
>>> mammoth = '''
... We have seen thee, queen of cheese,
... Lying quietly at your ease,
... Gently fanned by evening breeze,
... Thy fair form no flies dare seize.
...
... All gaily dressed soon you'll go
... To the great Provincial show,
... To be admired by many a beau
... In the city of Toronto.
...
... Cows numerous as a swarm of bees,
... Or as the leaves upon the trees,
... It did require to make thee please,
... And stand unrivalled, queen of cheese.
...
... May you not receive a scar as
... We have heard that Mr. Harris
... Intends to send you off as far as
... The great world's show at Paris.
...
... Of the youth beware of these,
... For some of them might rudely squeeze
... And bite your cheek, then songs or glees
... We could not sing, oh! queen of cheese.
...
... We'rt thou suspended from balloon,
... You'd cast a shade even at noon,
... Folks would think it was the moon
... About to fall and crush them soon.
... '''
```

17.5. Импортируйте модуль `re`, чтобы получить доступ к поддержке регулярных выражений в Python. Используйте функцию `re.findall()` для вывода на экран всех слов, которые начинаются с буквы c.

Мы определим переменную `pat` для шаблона и затем будем искать такой шаблон в строке `mammoth`:

```
>>> import re
>>> pat = r'\bc\w*'
>>> re.findall(pat, mammoth)
['cheese', 'city', 'cheese', 'cheek', 'could', 'cheese', 'cast', 'crush']
```

Элемент `\b` означает, что совпадение должно начинаться на границе слова. Используйте такую конструкцию, чтобы указать либо на начало, либо на конец слова. Литерал `c` — это первая буква в искомых словах. Элемент `\w` означает *любой символ слова* и включает буквы, цифры и символ подчеркивания (_).

Элемент * означает *ноль или больше* таких символов. В целом это выражение ищет слова, которые начинаются с c, включая 'c'. Если вы не использовали неинтерпретируемую строку (определения таких строк начинаются с символа r перед открывающей кавычкой), то Python воспримет \b как возврат на шаг и поиск потерпит неудачу:

```
>>> pat = '\bc\w*'
>>> re.findall(pat, mammoth)
[]
```

17.6. Найдите все четырехбуквенные слова, которые начинаются с буквы c.

```
>>> pat = r'\bc\w{3}\b'
>>> re.findall(pat, mammoth)
['city', 'cast']
```

Заключительный символ \b необходим, чтобы обозначить конец слова. В противном случае поиск вернет первые четыре буквы всех слов, которые начинаются с c и содержат не менее четырех букв:

```
>>> pat = r'\bc\w{3}'
>>> re.findall(pat, mammoth)
['chee', 'city', 'chee', 'chee', 'coul', 'chee', 'cast', 'crus']
```

17.7. Найдите все слова, которые заканчиваются на букву r.

Это упражнение с подвохом. Вот как получить все слова, которые заканчиваются на r:

```
>>> pat = r'\b\w*r\b'
>>> re.findall(pat,mammoth)
['your', 'fair', 'Or', 'scar', 'Mr', 'far', 'For', 'your', 'or']
```

Однако результаты будут не так хороши, если попробовать отыскать слова, которые заканчиваются на l:

```
>>> pat = r'\b\w*l\b'
>>> re.findall(pat,mammoth)
['All', 'll', 'Provincial', 'fall']
```

Что здесь делает буквосочетание ll? Паттерн \w совпадает только с буквами, цифрами и подчеркиваниями, но не с апострофами ASCII. В результате регулярное выражение совпало с буквосочетанием ll. Этот случай можно обработать, добавив апостроф в список символов, с которыми возможно совпадение. Наша первая попытка не работает:

```
>>> pat = r'\b[\w']*l\b'
   File "<stdin>", line 1
      pat = r'\b[\w']*l\b'
```

Python указывает на окрестности ошибки, но может потребоваться какое-то время, чтобы осознать: ошибка в том, что строка шаблона окружена такими же

апострофами — символами кавычки. Одно из решений проблемы — использовать управляющую последовательность с обратным слешем:

```
>>> pat = r'\b[\w\']*l\b'
>>> re.findall(pat, mammoth)
['All', "you'll", 'Provincial', 'fall']
```

Еще одно решение — окружить строку шаблона двойными кавычками:

```
>>> pat = r"\b[\w']*l\b"
>>> re.findall(pat, mammoth)
['All', "you'll", 'Provincial', 'fall']
```

17.8. Найдите все слова, которые содержат три гласные подряд.

Начиная с границы слова, любое количество символов *слова*, затем три гласные и далее любые символы, не являющиеся гласными, до конца слова:

```
>>> pat = r'\b[^aeiou]*[aeiou]{3}[^aeiou]*\b'
>>> re.findall(pat, mammoth)
['queen', 'quietly', 'beau\nIn', 'queen', 'squeeze', 'queen']
```

Выглядит правильно, за исключением строки `'beau\nIn'`. Мы выполнили поиск по строке `mammoth`, содержащей многострочный текст. Конструкция `[^aeiou]` совпадает с любыми символами, не являющимися гласными, включая `\n` (перенос строки, который отмечает конец текстовой строки). Нам нужно добавить еще кое-что в набор игнорируемых символов — `\s` совпадает с любыми символами пробелов, включая `\n`:

```
>>> pat = r'\b\w*[aeiou]{3}[^aeiou\s]\w*\b'
>>> re.findall(pat, mammoth)
['queen', 'quietly', 'queen', 'squeeze', 'queen']
```

На сей раз мы не нашли слово `beau`, поэтому внесем в шаблон еще одно исправление — совпадение с любым числом (даже нулем) не гласных после трех гласных. Наш предыдущий шаблон всегда совпадал с одним не гласным символом:

```
>>> pat = r'\b\w*[aeiou]{3}[^aeiou\s]*\w*\b'
>>> re.findall(pat, mammoth)
['queen', 'quietly', 'beau', 'queen', 'squeeze', 'queen']
```

Что это показывает? Что регулярные выражения могут многое, но зачастую их очень трудно написать правильно с первого раза.

Глава 18. Двоичные данные

18.1. Используйте метод `unhexlify()`, чтобы преобразовать шестнадцатеричную строку в переменной `gif` типа `bytes` (созданную объединением двух строк из-за необходимости уместить ее по ширине книжной страницы):

```
'47494638396101000100800000000000ffffff21f9' +
'0401000000002c000000000100010000020144003b'
```

```
>>> import binascii
>>> hex_str = '47494638396101000100800000000000ffffff21f9' + \
...       '0401000000002c000000000100010000020144003b'
>>> gif = binascii.unhexlify(hex_str)
>>> len(gif)
42
```

18.2. Байты, содержащиеся в переменной `gif`, определяют однопиксельный прозрачный GIF-файл. Этот формат — один из самых распространенных. Корректный файл в формате GIF начинается со строки `GIF89a`. Корректен ли этот файл?

```
>>> gif[:6] == b'GIF89a'
True
```

Обратите внимание: с помощью `b` мы можем указать, что строка состоит из байтов, а не из символов Юникода. Вы можете сравнить байты с байтами, но не байты с символами:

```
>>> gif[:6] == 'GIF89a'
False
>>> type(gif)
<class 'bytes'>
>>> type('GIF89a')
<class 'str'>
>>> type(b'GIF89a')
<class 'bytes'>
```

18.3. Ширина изображения GIF — это 16-битное целое число с обратным порядком байтов, которое начинается со смещения 6 байт. Его высота имеет такой же размер и начинается со смещения 8 байт. Извлеките и выведите на экран эти значения из переменной `gif`. Равны ли они `1`?

```
>>> import struct
>>> width, height = struct.unpack('<HH', gif[6:10])
>>> width, height
(1, 1)
```

Глава 19. Дата и время

19.1. Запишите текущие дату и время как строку в текстовый файл `today.txt`.

```
>>> from datetime import date
>>> now = date.today()
>>> now_str = now.isoformat()
```

```
>>> with open('today.txt', 'wt') as output:
...         print(now_str, file=output)
>>>
```

Когда я запустил этот фрагмент кода, в файле `today.txt` увидел следующее:

```
2025-04-23
```

Вместо `print` можно было бы использовать функцию `write()`, например `output.write(now_str)`. Функция `print` добавляет символ перевода строки в конец.

19.2. Прочтите текстовый файл `today.txt` и поместите данные в строку `today_string`.

```
>>> with open('today.txt', 'rt') as input:
...         today_string = input.read()
...
>>> today_string
'2025-04-23\n'
```

19.3. Проанализируйте дату из строки `today_string`.

```
>>> fmt = '%Y-%m-%d\n'
>>> datetime.strptime(today_string, fmt)
datetime.datetime(2025, 4, 23, 0, 0)
```

Если вы записали этот символ новой строки в файл, то вам нужно, чтобы он совпал со строкой формата.

19.4. Создайте объект `date` с датой вашего рождения.

Предположим, вы родились 14 августа 1982 года:

```
>>> my_day = date(1982, 8, 14)
>>> my_day
datetime.date(1982, 8, 14)
```

19.5. В какой день недели вы родились?

```
>>> my_day.weekday()
5
>>> my_day.isoweekday()
6
```

Функция `weekday()` возвращает `0` для понедельника и `6` — для воскресенья. Функция `isoweekday()` возвращает `1` для понедельника и `7` — для воскресенья. Поэтому искомый день — суббота.

19.6. Когда вам будет (или уже было) 10 000 дней от роду?

```
>>> from datetime import timedelta
>>> party_day = my_day + timedelta(days=10000)
>>> party_day
datetime.date(2009, 12, 30)
```

Если это был ваш день рождения, то вы, возможно, пропустили еще один повод повеселиться.

Глава 20. Файлы

20.1. Выведите на экран список файлов, находящихся в текущем каталоге.

Если ваш текущий каталог называется ohmy и содержит три файла с именами, соответствующими названиям животных, то код может выглядеть так:

```
>>> import os
>>> os.listdir('.')
['bears', 'lions', 'tigers']
```

20.2. Выведите на экран список файлов, находящихся в родительском каталоге.

Если родительский каталог содержит два файла и текущий каталог ohmy, то код может выглядеть так:

```
>>> import os
>>> os.listdir('..')
['ohmy', 'paws', 'whiskers']
```

20.3. Присвойте строку `'This is a test of the emergency text system'` переменной `test1` и запишите переменную `test1` в файл `test.txt`.

```
>>> test1 = 'This is a test of the emergency text system'
>>> len(test1)
43
```

Вот как можно сделать это с помощью функций `open`, `write` и `close`:

```
>>> outfile = open('test.txt', 'wt')
>>> outfile.write(test1)
43
>>> outfile.close()
```

Или можно использовать `with` и избавиться от вызова `close` (Python сделает это за вас):

```
>>> with open('test.txt', 'wt') as outfile:
...       outfile.write(test1)
...
43
```

20.4. Откройте файл `test.txt` и прочитайте его содержимое в строку `test2`. Совпадают ли строки `test1` и `test2`?

```
>>> with open('test.txt', 'rt') as infile:
...       test2 = infile.read()
...
>>> len(test2)
43
>>> test1 == test2
True
```

Глава 21. Данные во времени: конкурентность

21.1. Используйте модуль `multiprocessing`, чтобы создать три отдельных процесса, как показано в примере П.13. Заставьте каждый из них ждать случайное количество секунд (от нуля до одной), вывести текущее время и завершить работу.

Пример П.13. multi_times.py

```
import multiprocessing

def now(seconds):
      from datetime import datetime
      from time import sleep
      sleep(seconds)
      print('wait', seconds, 'seconds, time is', datetime.utcnow())

if __name__ == '__main__':
      import random
      for n in range(3):
      seconds = random.random()
      proc = multiprocessing.Process(target=now, args=(seconds,))
      proc.start()

$ python multi_times.py
wait 0.10720361113059229 seconds, time is 2019-07-24 00:19:23.951594
wait 0.5825144002370065 seconds, time is 2019-07-24 00:19:24.425047
wait 0.6647690569029477 seconds, time is 2019-07-24 00:19:24.509995
```

Глава 22. Данные в пространстве: сети

22.1. Используйте простой сокет и реализуйте сервис, сообщающий текущее время. Когда клиент отправляет на сервер строку `'time'`, тот должен вернуть текущие дату и время как строку ISO.

В примере П.14 показан один из вариантов того, как можно написать сервер `udp_time_server.py`. Обратите внимание: если использовать `datetime.datetime.utcnow()` для получения текущего времени в формате UTC, то интерпретатор предупредит, что эта функция устарела. Она будет удалена в одной из будущих версий Python. Вместо нее рекомендуется задействовать `datetime.datetime.now(datetime.UTC)`.

Пример П.14. udp_time_server.py

```
import datetime
import socket

address = ('localhost', 6789)
max_size = 4096
print('Starting the server at', datetime.datetime.now(datetime.UTC))
print('Waiting for a client to call.')
server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server.bind(address)
while True:
      data, client_addr = server.recvfrom(max_size)
      if data == b'time':
            now = str(datetime.datetime.now(datetime.UTC))
            data = now.encode('utf-8')
            server.sendto(data, client_addr)
            print('Server sent', data)
server.close()
```

А в примере П.15 — клиент `udp_time_client.py`.

Пример П.15. udp_time_client.py

```
import socket
from datetime import datetime
from time import sleep

address = ('localhost', 6789)
max_size = 4096

print('Starting the client at', datetime.now())
client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
while True:
      sleep(5)
      client.sendto(b'time', address)
      data, server_addr = client.recvfrom(max_size)
      print('Client read', data)
client.close()
```

Я поместил вызов `sleep(5)` в начало цикла на стороне клиента, чтобы замедлить обмен данными. Запустите сервер в одном окне:

```
$ python udp_time_server.py
Starting the server at 2025-04-23 23:14:12.095702
Waiting for a client to call.
```

Запустите клиент в другом окне:

```
$ python udp_time_client.py
Starting the client at 2025-04-23 23:14:39.915397
```

Через 5 секунд в обоих окнах начнут появляться сообщения. Так выглядят первые три строки на стороне сервера:

```
Server sent b'2025-04-24 04:14:44.924383'
Server sent b'2025-04-24 04:14:49.929807'
Server sent b'2025-04-24 04:14:54.931491'
```

А так — первые три строки на стороне клиента:

```
Client read b'2025-04-24 04:14:44.924383'
Client read b'2025-04-24 04:14:49.929807'
Client read b'2025-04-24 04:14:54.931491'
```

Обе программы будут работать до бесконечности, поэтому их следует завершить вручную.

22.2. Реализуйте тот же сервис с помощью сокетов ZeroMQ `REQ` и `REP` (примеры П.16 и П.17).

Пример П.16. zmq_time_server.py

```
import zmq
import datetime

host = '127.0.0.1'
port = 6789
context = zmq.Context()
server = context.socket(zmq.REP)
server.bind("tcp://%s:%s" % (host, port))
now = datetime.datetime.now(datetime.UTC)
print('Server started at', now)
while True:
      # Ждать запроса от клиента
      message = server.recv()
      if message == b'time':
            now = datetime.datetime.now(datetime.UTC)
            reply = str(now)
            server.send(bytes(reply, 'utf-8'))
            print('Server sent', reply)
```

Пример П.17. zmq_time_client.py

```
import zmq
import datetime
from time import sleep

host = '127.0.0.1'
port = 6789
context = zmq.Context()
client = context.socket(zmq.REQ)
client.connect("tcp://%s:%s" % (host, port))
now = datetime.datetime.now(datetime.UTC)
```

```
print('Client started at', now)
while True:
      sleep(5)
      request = b'time'
      client.send(request)
      reply = client.recv()
      print("Client received %s" % reply)
```

При использовании простых сокетов приходилось сначала запускать сервер, а потом клиент. При использовании сокетов ZeroMQ порядок запуска неважен:

```
$ python zmq_time_server.py
Server started at 2025-04-24 04:32:31.014781+00:00

$ python zmq_time_client.py
Client started at 2025-04-24 04:32:33.877329+00:00
```

Через 15 секунд должны появиться сообщения на стороне сервера:

```
Server sent 2025-04-24 04:32:38.882717+00:00
Server sent 2025-04-24 04:32:43.888701+00:00
Server sent 2025-04-24 04:32:48.894584+00:00
```

В ответ должны появиться строки на стороне клиента:

```
Client received b'2025-04-24 04:32:38.882717+00:00'
Client received b'2025-04-24 04:32:43.888701+00:00'
Client received b'2025-04-24 04:32:48.894584+00:00'
```

22.3. Попробуйте сделать то же самое с помощью XML RPC.

В примере П.18 приводится серверный код.

Пример П.18. xmlrpc_time_server.py

```
from xmlrpc.server import SimpleXMLRPCServer

def now():
      import datetime
      data = str(datetime.datetime.now(datetime.UTC))
      print('Server sent', data)
      return data

server = SimpleXMLRPCServer(("localhost", 6789))
server.register_function(now, "now")
server.serve_forever()
```

А в примере П.19 дан соответствующий клиентский код.

Пример П.19. xmlrpc_time_client.py

```
import xmlrpc.client
from time import sleep

proxy = xmlrpc.client.ServerProxy("http://localhost:6789/")
while True:
      sleep(5)
      data = proxy.now()
      print('Client received', data)
```

Запустим сервер:

```
$ python xmlrpc_time_server.py
```

Запустим клиент:

```
$ python xmlrpc_time_client.py
```

Подождите примерно 15 секунд. Так выглядят первые три строки на стороне сервера:

```
Server sent 2025-04-24 04:39:14.842428+00:00
127.0.0.1 - - [23/Apr/2025 23:39:14] "POST / HTTP/1.1" 200 -
Server sent 2025-04-24 04:39:19.847258+00:00
127.0.0.1 - - [23/Apr/2025 23:39:19] "POST / HTTP/1.1" 200 -
Server sent 2025-04-24 04:39:24.851759+00:00
127.0.0.1 - - [23/Apr/2025 23:39:24] "POST / HTTP/1.1" 200 -
```

А так — первые три строки на стороне клиента:

```
Client received 2025-04-24 04:39:14.842428+00:00
Client received 2025-04-24 04:39:19.847258+00:00
Client received 2025-04-24 04:39:24.851759+00:00
```

22.4. Возможно, вы видели эпизод телесериала *I Love Lucy*, в котором Люси и Этель работают на шоколадной фабрике (это классика). Парочка стала отставать, когда линия конвейера, направлявшая к ним на обработку конфеты, начала ускоряться. Напишите симуляцию, которая отправляет разные типы конфет в список Redis, и клиент `Lucy`, выполняющий блокирующую операцию извлечения элемента из списка. Клиенту нужно 0,5 секунды, чтобы обработать одну конфету. Выведите на экран время и тип каждой конфеты, которую получит `Lucy`, а также количество необработанных конфет.

Симуляция `redis_choc_supply.py` (пример П.20) передает бесконечное количество конфет.

Пример П.20. redis_choc_supply.py

```
import redis
import random
from time import sleep

conn = redis.Redis()
varieties = ['truffle', 'cherry', 'caramel', 'nougat']
conveyor = 'chocolates'
while True:
      seconds = random.random()
      sleep(seconds)
      piece = random.choice(varieties)
      conn.rpush(conveyor, piece)
```

Клиент `redis_lucy.py` может выглядеть так, как показано в примере П.21.

Пример П.21. redis_lucy.py

```
import redis
from datetime import datetime
from time import sleep
```

```
conn = redis.Redis()
timeout = 10
conveyor = 'chocolates'
while True:
      sleep(0.5)
      msg = conn.blpop(conveyor, timeout)
      remaining = conn.llen(conveyor)
      if msg:
            piece = msg[1]
            print('Lucy got a', piece, 'at', datetime.utcnow(),
            ', only', remaining, 'left')
```

Запустите сначала сервер Redis, а потом два предыдущих сценария в любом порядке. Поскольку Люси требуется полсекунды для обработки каждой конфеты, а они появляются в среднем каждые полсекунды, ситуация начинает напоминать гонку. Чем раньше вы запустите конвейер, тем сложнее придется Люси:

```
$ python redis_choc_supply.py &

$ python redis_lucy.py
Lucy got a b'truffle' at 2025-06-04 21:58:25.088134 , only 22 left
Lucy got a b'nougat' at 2025-06-04 21:58:25.589885 , only 22 left
Lucy got a b'cherry' at 2025-06-04 21:58:26.092401 , only 21 left
Lucy got a b'cherry' at 2025-06-04 21:58:26.594963 , only 21 left
Lucy got a b'nougat' at 2025-06-04 21:58:27.097996 , only 20 left
Lucy got a b'caramel' at 2025-06-04 21:58:27.600008 , only 21 left
Lucy got a b'truffle' at 2025-06-04 21:58:28.102135 , only 22 left
Lucy got a b'cherry' at 2025-06-04 21:58:28.604621 , only 21 left
Lucy got a b'nougat' at 2025-06-04 21:58:29.107136 , only 22 left
Lucy got a b'cherry' at 2025-06-04 21:58:29.609579 , only 22 left
Lucy got a b'cherry' at 2025-06-04 21:58:30.111380 , only 23 left
Lucy got a b'truffle' at 2025-06-04 21:58:30.613568 , only 23 left
Lucy got a b'caramel' at 2025-06-04 21:58:31.115780 , only 24 left
Lucy got a b'cherry' at 2025-06-04 21:58:31.618605 , only 23 left
Lucy got a b'truffle' at 2025-06-04 21:58:32.121005 , only 23 left
Lucy got a b'cherry' at 2025-06-04 21:58:32.623156 , only 23 left
Lucy got a b'cherry' at 2025-06-04 21:58:33.125025 , only 23 left
Lucy got a b'cherry' at 2025-06-04 21:58:33.626977 , only 23 left
Lucy got a b'cherry' at 2025-06-04 21:58:34.128611 , only 23 left
```

Бедная Люси.

Глава 23. Данные в коробке: персистентные хранилища

23.1. Сохраните следующие несколько строк в файл `books.csv`. Обратите внимание на то, что если поля разделены запятыми, то поля, содержащие запятые, нужно заключить в кавычки:

```
author,book
J R R Tolkien,The Hobbit
Lynne Truss,"Eats, Shoots & Leaves"
```

```
>>> text = '''author,book
... J R R Tolkien,The Hobbit
... Lynne Truss, "Eats, Shoots & Leaves"
... '''
>>> with open('test.csv', 'wt') as outfile:
...       outfile.write(text)
...
73
```

23.2. Используйте модуль `csv` и его метод `DictReader`, чтобы прочитать содержимое файла `books.csv` в переменную `books`. Выведите на экран содержимое переменной `books`. Обработал ли метод `DictReader` кавычки и запятые в заголовке второй книги?

```
>>> with open('books.csv', 'rt') as infile:
...       books = csv.DictReader(infile)
...       for book in books:
...             print(book)
...
{'book': 'The Hobbit', 'author': 'J R R Tolkien'}
{'book': 'Eats, Shoots & Leaves', 'author': 'Lynne Truss'}
```

23.3. Создайте CSV-файл `books.csv` и запишите в него в следующие строки:

```
title,author,year
The Weirdstone of Brisingamen,Alan Garner,1960
Perdido Street Station,China Miéville,2000
Thud!,Terry Pratchett,2005
The Spellman Files,Lisa Lutz,2007
Small Gods,Terry Pratchett,1992
```

```
>>> text = '''title,author,year
... The Weirdstone of Brisingamen,Alan Garner,1960
... Perdido Street Station,China Miéville,2000
... Thud!,Terry Pratchett,2005
... The Spellman Files,Lisa Lutz,2007
... Small Gods,Terry Pratchett,1992
... '''
>>> with open('books2.csv', 'wt') as outfile:
...       outfile.write(text)
...
201
```

23.4. Используйте модуль `sqlite3`, чтобы создать базу данных SQLite `books.db` и таблицу `books` со следующими полями: `title` (текст), `author` (текст) и `year` (целое значение).

```
>>> import sqlite3
>>> db = sqlite3.connect('books.db')
>>> curs = db.cursor()
```

```
>>> curs.execute('''create table book (title text, author text, year int)''')
<sqlite3.Cursor object at 0x1006e3b90>
>>> db.commit()
```

23.5. Прочитайте данные из файла `books2.csv` и добавьте их в таблицу `book`.

```
>>> import csv
>>> import sqlite3
>>> ins_str = 'insert into book values(?, ?, ?)'
>>> with open('books.csv', 'rt') as infile:
...     books = csv.DictReader(infile)
...     for book in books:
...         curs.execute(ins_str, (book['title'], book['author'], book['year']))
...
<sqlite3.Cursor object at 0x1007b21f0>
<sqlite3.Cursor object at 0x1007b21f0>
<sqlite3.Cursor object at 0x1007b21f0>
<sqlite3.Cursor object at 0x1007b21f0>
<sqlite3.Cursor object at 0x1007b21f0>
>>> db.commit()
```

23.6. Прочитайте и выведите на экран содержимое столбца `title` таблицы `book` в алфавитном порядке.

```
>>> sql = 'select title from book order by title asc'
>>> for row in db.execute(sql):
...         print(row)
...
('Perdido Street Station',)
('Small Gods',)
('The Spellman Files',)
('The Weirdstone of Brisingamen',)
('Thud!',)
```

Чтобы вывести на экран значение `title` без элементов, свойственных кортежам (круглых скобок и запятых), попробуйте следующее:

```
>>> for row in db.execute(sql):
...         print(row[0])
...
Perdido Street Station
Small Gods
The Spellman Files
The Weirdstone of Brisingamen
Thud!
```

Если хотите проигнорировать начальное слово `'The'` в заголовках, то вам нужно написать еще одну строку SQL:

```
>>> sql = '''select title from book order by
... case when (title like "The %")
... then substr(title, 5)
... else title end'''
```

```
>>> for row in db.execute(sql):
...       print(row[0])
...
Perdido Street Station
Small Gods
The Spellman Files
Thud!
The Weirdstone of Brisingamen
```

23.7. Прочитайте и выведите на экран содержимое всех столбцов таблицы book в порядке публикации.

```
>>> for row in db.execute('select * from book order by year'):
...       print(row)
...
('The Weirdstone of Brisingamen', 'Alan Garner', 1960)
('Small Gods', 'Terry Pratchett', 1992)
('Perdido Street Station', 'China Miéville', 2000)
('Thud!', 'Terry Pratchett', 2005)
('The Spellman Files', 'Lisa Lutz', 2007)
```

Чтобы вывести поля, имеющиеся в каждой записи, просто разделите их запятыми и пробелами:

```
>>> for row in db.execute('select * from book order by year'):
...       print(*row, sep=', ')
...
The Weirdstone of Brisingamen, Alan Garner, 1960
Small Gods, Terry Pratchett, 1992
Perdido Street Station, China Miéville, 2000
Thud!, Terry Pratchett, 2005
The Spellman Files, Lisa Lutz, 2007
```

23.8. Используйте модуль SQLAlchemy, чтобы подключиться к базе данных `sqlite3 books.db`, созданной в упражнении 23.4. Как и в упражнении 23.6, прочитайте и выведите на экран содержимое столбца `title` таблицы book в алфавитном порядке.

```
>>> import sqlalchemy
>>> conn = sqlalchemy.create_engine('sqlite:///books.db')
>>> sql = 'select title from book order by title asc'
>>> rows = conn.execute(sql)
>>> for row in rows:
...       print(row)
...
('Perdido Street Station',)
('Small Gods',)
('The Spellman Files',)
('The Weirdstone of Brisingamen',)
('Thud!',)
```

23.9. Установите сервер Redis и библиотеку `redis` (командой `pip install redis`) на свой компьютер. Создайте хеш Redis `test`, содержащий поля `count (1)` и `name ('Fester Bestertester')`. Выведите все поля хеша `test`.

```
>>> import redis
>>> conn = redis.Redis()
>>> conn.delete('test')
1
>>> conn.hmset('test', {'count': 1, 'name': 'Fester Bestertester'})
True
>>> conn.hgetall('test')
{b'name': b'Fester Bestertester', b'count': b'1'}
```

23.10. Увеличьте поле `count` хеша `test` и выведите его на экран.

```
>>> conn.hincrby('test', 'count', 3)
4
>>> conn.hget('test', 'count')
b'4'
```

Глава 24. Всемирная паутина

24.1. Если вы еще не установили Flask, то сделайте это сейчас, чтобы получить возможность установить Werkzeug, Jinja2 и, возможно, другие пакеты.

24.2. Создайте скелет сайта с помощью веб-сервера Flask. Убедитесь, что сервер доступен по адресу `localhost` на стандартном порте 5000. Если ваш компьютер уже задействует этот порт для чего-то еще, то воспользуйтесь другим портом.

В примере П.22 показано содержимое файла `flask1.py`.

Пример П.22. flask1.py

```
from flask import Flask

app = Flask(__name__)

app.run(port=5000, debug=True)
```

Поехали:

```
$ python flask1.py
* Running on http://127.0.0.1:5000/
* Restarting with reloader
```

24.3. Добавьте функцию `home()` для обработки запросов к главной странице. Пусть она возвращает строку `It's alive!`.

Как нам назвать этот файл, `flask2.py` (пример П.23)?

Пример П.23. flask2.py

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
      return "It's alive!"

app.run(debug=True)
```

Запустим сервер:

```
$ python flask2.py
* Running on http://127.0.0.1:5000/
* Restarting with reloader
```

Наконец, получим доступ к главной странице с помощью браузера, HTTP-программы командной строки, такой как `curl`, или `wget`, или даже `telnet`:

```
$ curl http://localhost:5000/
It's alive!
```

24.4. Создайте в файле `home.html` шаблон `Jinja2`, содержащий следующий код:

```
<html>
<head>
<title>It's alive!</title>
<body>
I'm of course referring to {{thing}}, which is {{height}} feet tall and {{color}}.
</body>
</html>
```

Создайте каталог `templates` и файл `home.html` с приведенным кодом. Если ваш сервер Flask все еще работает после запуска предыдущих примеров, то он обнаружит, что файл обновился, и перезапустится.

24.5. Модифицируйте функцию `home()` так, чтобы она использовала шаблон `home.html`. Передайте ей три параметра, полученные из запроса `GET`: `thing`, `height` и `color`.

Содержимое файла `flask3.py` показано в примере П.24.

Пример П.24. flask3.py

```
from flask import Flask, request, render_template

app = Flask(__name__)

@app.route('/')
def home():
      thing = request.values.get('thing')
      height = request.values.get('height')
      color = request.values.get('color')
```

```
      return render_template('home.html',
            thing=thing, height=height, color=color)
app.run(debug=True)
```

Перейдите в клиенте по адресу:

```
http://localhost:5000/?thing=Octothorpe&height=7&color=green
```

Вы должны увидеть вот что:

```
I'm of course referring to Octothorpe, which is 7 feet tall and green.
```

Глава 25. Наука о данных

25.1. Установите DuckDB и заново создайте базу данных `zoo` из подраздела «SQLite» в главе 23 (пример П.25).

Пример П.25. duck.py

```
import duckdb

conn = duckdb.connect(database = ":memory:")
conn.execute('''create table zoo
       (critter varchar(20) primary key,
      count int,
      damages float)''')
ins = 'insert into zoo (critter, count, damages) values(?, ?, ?)'
conn.execute(ins, ('duck', 5, 0.0))
conn.execute(ins, ('bear', 2, 1000.0))
conn.execute(ins, ('weasel', 1, 2000.0))
conn.execute('select * from zoo')
print(conn.fetchall())
conn.execute('select * from zoo order by count')
print(conn.fetchall())
conn.execute('select * from zoo order by count desc')
print(conn.fetchall())
conn.execute('''select * from zoo where
      damages = (select max(damages) from zoo)''')
print(conn.fetchall())
```

```
$ python duck.py
[(duck, 5, 0.0), (bear, 2, 1000.0), (weasel, 1, 2000.0)]
[(weasel, 1, 2000.0), (bear, 2, 1000.0), (duck, 5, 0.0)]
[(duck, 5, 0.0), (bear, 2, 1000.0), (weasel, 1, 2000.0)]
[(weasel, 1, 2000.0)]
```

Глава 26. Искусственный интеллект

Ранее, в разделе «Рабочие примеры с использованием Ollama» главы 26, я привел ряд примеров, которые оказались не совсем рабочими. Выберите любую другую модель ИИ (с открытым исходным кодом или коммерческую, локальную или хостинговую) и попробуйте с ее помощью выполнить следующие задания.

Используйте модель ChatGPT, которая нынче хорошо известна.

26.1. Подсчитайте количество букв r в слове strawberry.

> *Запрос:* Сколько раз буква r встречается в слове strawberry?
>
> *ChatGPT:* В слове **strawberry две буквы** r.

Ура. Давайте проверим, сможет ли ChatGPT написать код, вычисляющий правильный ответ.

> *Запрос:* Напиши программу на Python для подсчета количества вхождений буквы r в слово strawberry.
>
> *ChatGPT:* Конечно! Вот простая программа на Python для подсчета количества букв r в слове strawberry:
>
> ```
> # Определение слова
> word = "strawberry"
> # Подсчет количества вхождений буквы r
> count = word.count('r')
> # Вывод результата
> print(f"Буква 'r' в слове 'strawberry' встречается {count} раз.")
> ```
>
> # Объяснение
>
> - Для подсчета количества вхождений буквы r в слово strawberry используется строковый метод `count()`.
> - Метод `count()` возвращает целое число — количество вхождений указанной подстроки (в данном случае r).
>
> # Вывод:
>
> ```
> Буква 'r' встречается в слове 'strawberry' 2 раза.
> ```
>
> Дайте мне знать, если вам нужны дополнительные разъяснения или изменения!

Странно! ChatGPT генерирует правильный код на Python, который возвращает `3`, но говорит, что он возвращает `2`. Я повторно отправил исходный вопрос в ChatGPT несколько дней спустя.

Запрос: Напиши программу на Python для подсчета количества вхождений буквы r в слово strawberry.

На этот раз модель дала правильный ответ.

ChatGPT: Буква r встречается в слове strawberry три раза.

Интересно отметить, что теперь модель ChatGPT использовала цифру 3 вместо слова «два» для обозначения количества букв.

26.2. Найдите определение лимерика.

Запрос: Что такое лимерик?

ChatGPT: Лимерик — это форма короткого юмористического стихотворения, состоящего из пяти строк, с определенной рифмовкой и ритмической структурой. Вот основные характерные черты лимерика.

Структура: пять строк.

Структура рифмы: AABBA.

Первая, вторая и пятая строки рифмуются между собой и обычно длиннее третьей и четвертой (часто имеют длину 8–9 слогов).

Третья и четвертая строки рифмуются между собой и обычно короче остальных (5–6 слогов).

Тон: беззаботный, игривый, часто абсурдный или бессмысленный.

Может включать игру слов, каламбуры и даже пикантные шутки.

Пример:

```
Кудреватый старик из Перу
Всё не знал, чем заняться в миру;
Неуклюж, как медведь,
Выдрал кудри на треть
Лысоватый старик из Перу.
```

Дайте мне знать, если вам нужна помощь в написании собственного лимерика!

Отлично!

26.3. Напишите лимерик, выбрав тему по своему вкусу.

Запрос: Напиши лимерик. Тему выбери сам.

ChatGPT: Конечно! Вот лимерик про кота на Луне:

```
Лунный кот по имени Сэр Орбитальный Пух
Решил, что космос — не шутка, а дух.
Соорудил он из чаши
Путь к звездной фазе,
И вот он — Кот — Вселенский Друг!
```

Хотите другой стиль — глупый, жуткий или романтичный?

И снова успех!

26.4. Напишите на Python функцию `utc_now_str()`, которая возвращает текущие дату и время UTC в виде строки.

ChatGPT: Конечно! функцию на Python, которая возвращает текущие дату и время UTC в виде строки:

```
from datetime import datetime

def utc_now_str():
   # Возвращает текущее время UTC в виде строки
   return datetime.utcnow().strftime('%Y-%m-%d %H:%M:%S')

# Пример использования
print(utc_now_str())
```

Эта функция использует метод `datetime.utcnow()` для получения текущего времени в формате UTC, а затем форматирует его как строку в формате ГГГГ-ММ-ДД ЧЧ:ММ:СС.

Дайте мне знать, если вам потребуются какие-либо изменения!

Модель ChatGPT сгенерировала вполне рабочую функцию, но она не знала, что `utcnow()` исчезнет в будущем.

26.5. Напишите генератор Python, возвращающий целые числа от 1 до 5.

ChatGPT: Вот как можно написать генератор на Python:

```
gen = (x for x in range(1, 6))

# Пример использования: вывод сгенерированных значений
for num in gen:
      print(num)
```

Этот код создает генератор, который при использовании в цикле выдает целые числа от 1 до 5.

Дайте мне знать, если вам нужны подробности!

Снова верный ответ.

26.6. Повторите пример 1.5.

Весь следующий текст в рамке — это ответ ChatGPT.

Запрос: Напиши программу на Python, которая обращается к Wayback Machine на сайте Internet Archive. Программа должна запрашивать у пользователя адрес сайта и дату, а затем запрашивать у Wayback Machine (с помощью библиотеки

Requests) архивную версию указанного сайта, наиболее близкую к указанной дате. Если запрашиваемая версия найдена, выведи ее URL и открой его в браузере по умолчанию. В противном случае выведи сообщение об ошибке.

ChatGPT: Вот полная программа на Python, которая делает то, что вы описываете:

```
import requests
import webbrowser

def main():
      print("Wayback Machine Archive Finder")
      domain = input("Enter a domain (e.g. example.com): ").strip()
      date = input("Enter a date (YYYYMMDD): ").strip()

      url = (f"http://archive.org/wayback/available?" +
            f"url={domain}&timestamp={date}")

try:
      response = requests.get(url, timeout=10)
      response.raise_for_status()
      data = response.json()
            if "archived_snapshots" in data and \
                  "closest" in data["archived_snapshots"]:
                  archived_url = data["archived_snapshots"]["closest"]["url"]
                  print(f"Archived version found: {archived_url}")
                  webbrowser.open(archived_url)
            else:
                print("No archived version found for that domain and date.")

      except requests.RequestException as e:
            print(f"Error accessing Wayback Machine API: {e}")

if __name__ == "__main__":
      main()
```

Как она работает

Запрашивает у пользователя адрес сайта и дату в формате ГГГГММДД.

Посылает запрос к Wayback Machine Availability API (http://archive.org/wayback/available).

Если существует архивный снимок, то выводит URL и открывает его в браузере по умолчанию.

Если ничего не найдено, выводится сообщение об ошибке.

Ошибки (сеть, неверный ответ и т. д.) отслеживаются и отображаются.

Хотите добавить поддержку случаев, когда пользователь вводит только год или год + месяц (например, 2020 или 202003) вместо полного формата ГГГГММДД?

Этот код действительно хорош! Кстати, он гораздо короче первоначального, который ChatGPT вернула несколько месяцев назад.

26.7. Повторите пример 24.3.

Весь следующий текст в рамке — это ответ ChatGPT.

Запрос: Запроси у Internet Archive API фильм под названием Eegah. Выведи первые десять совпадений, выбери одно случайное и отобрази в браузере.

ChatGPT: Вот небольшой скрипт на Python, который делает именно это — запрашивает в интернет-архиве элементы с заголовком Eegah, выводит список первых десяти совпадений, выбирает одно наугад и открывает страницу с описанием в браузере по умолчанию:

```
#!/usr/bin/env python3
import requests
import random
import webbrowser
import sys

SEARCH_URL = "https://archive.org/advancedsearch.php"

def main():
    # Запрос: название "Eegah" и тип — фильм
    params = {
        "q": 'title: "Eegah" AND '+
             'mediatype:(movies OR television OR "moving image")',
        "fl[]": ["identifier", "title", "year", "date",
             "publicdate", "mediatype"],
        "rows": 10,                  # извлечь первые 10
        "page": 1,
        "output": "json"
    }

    try:
        r = requests.get(SEARCH_URL, params=params, timeout=20)
        r.raise_for_status()
    except requests.RequestException as e:
        print(f"Error talking to Internet Archive: {e}")
        sys.exit(1)
    data = r.json()
    docs = data.get("response", {}).get("docs", [])
    if not docs:
        print('No matches found for title "Eegah".')
        sys.exit(0)

    # Показать первые 10 совпадений
    print("First matches:")
    for i, d in enumerate(docs, start=1):
        year = d.get("year") or \
               (d.get("date") or d.get("publicdate") or "")[:4]
```

```
            title = d.get("title", "(no title)")
            mediatype = d.get("mediatype", "")
            print(f"{i:2d}. {title} ({year}) [{mediatype}] ->" +
                  f"https://archive.org/details/{d.get('identifier')}")

      # Выбрать одно наугад
      choice = random.choice(docs)
      chosen_url = f"https://archive.org/details/{choice['identifier']}"
      print("\nRandomly chosen item:")
      print(f"{choice.get('title','(no title)')} -> {chosen_url}")

      # Открыть в браузере по умолчанию
      try:
            webbrowser.open(chosen_url)
      except Exception as e:
            print(f"Could not open browser automatically: {e}")

if __name__ == "__main__":
      main()
```

Как использовать

1. Сохранить в файле `eegah_ia_search.py`.
2. Запустить командой `python3 eegah_ia_search.py`.

Эта функция выведет на экран первые десять совпадений (если таковые имеются), а затем наугад откроет одно из них в вашем браузере.

Как и в предыдущем упражнении, модель сгенерировала работоспособный код, вдобавок более компактный, чем код, который ChatGPT вернула несколько месяцев назад. Интересно, мы уже достигли сингулярности или нужно подождать еще несколько недель?

Глава 27. Производительность

27.1. Сложите числа от 1 до 100 следующими способами и засеките время (пример П.26):

- используя цикл `for`;
- задействуя пары чисел для вычисления суммы. Подсказка: найдите рассказ о том, как Гаусс, еще учась в школе, мгновенно вычислил сумму чисел от 1 до 100;
- используя прямую формулу на основе пар, чтобы получить сумму вычислением единственного выражения;

Есть ли в Python встроенная функция для суммирования таких последовательностей чисел?

Пример П.26. times.py

```
import time

def use_for():
      s = 0
      for n in range(1, 101):
            s += n
      return s

def use_pairs():
      # 1+100, 2+99, ... 50+51
      s = 0
      for n in range(1, 51):
            s += n + (101 - n) # or just 101
      return s

def use_formula():
      s = 50 * 101
      return s

def use_sum():
      return sum(range(1, 101))

def timer(func):
      t1 = time.time()
      s = func()
      t2 = time.time()
      print(f"{func.__name__:20}", s, f"{t2-t1:>20}")

timer(use_for)
timer(use_pairs)
timer(use_formula)
timer(use_sum)
```

```
$ python times.py
use_for                5050  7.867813110351562e-06
use_pairs              5050  5.4836273193359375e-06
use_formula            5050  2.384185791015625e-07
use_sum                5050  1.6689300537109375e-06
```