Занятие 12

List comprehension

Dictionary comprehension

Начнем с ...

- 1. Разминка (Что напечатает?)
- 2. Обсуждение домашнего задания

Что напечатает?

```
a = [1, 2, 3, 4]
b = a
a = a + [5, 6, 7, 8]

c = [1, 2, 3, 4]
d = c
c += [5, 6, 7, 8]

print(a, b)
print(c, d)
```

Задача 11-1

- Каждый третий четверг каждого месяца билеты в Эрмитаж бесплатны.
- Напечатайте список дат в 2023 году, когда вход бесплатен.

Задача 11-2

- Дан файл с расширением .csv, содержащий в каждой строчке следующую информацию: номер, фамилия, имя, компания, зарплата.
- Создайте Эксельный файл, в который перенесите эту информацию, предварительно отсортировав этот список по компании, по фамилии и имени.
- В конце списка добавьте строку: ИТОГО и суммарное значение всех зарплат.

Задача 11-3

- Напишите функцию, которая переводит арабские числа в римские.
- Например: 2023 ->MMXXIII

Time, Datetime, Calendar

- Time – удобен для оценки длительности программы, нахождения самых медленных и неэффективных ее частей
- Datetime – огромный набор классов и функций для решения разнообразных задач с датами и временем
- Calendar – модуль для работы с календарями

Некоторые функции модуля time

##	Функция	Что делает
1	import time	Импортирует модуль
2	time.time()	Текущее время в секундах
3	time.ctime()	Текущее время
4	time.ctime(t)	Дата и время из количества секунд t
5	t = (2019, 12, 7, 14, 30, 30, 5, 341, 0) time.asctime(t)	Дата и время на основании кортежа данных
6	t = (2019, 12, 7, 14, 30, 30, 5, 341, 0) local_time = time.mktime(t)	Обратная функция
7	time.sleep()	Приостанавливает выполнение программы на … секунд

Модуль datetime.timedelta

- Класс **datetime.timedelta** - разница между двумя моментами времени, с точностью до микросекунд.

- import datetime, locale
- locale.setlocale(locale.LC_ALL, 'ru')
- a = datetime.datetime.today() + datetime.timedelta(days = 14)
- print(a)
- c = datetime.timedelta(days = 1)
- for _ in range(10):
- a += c
- print(a.strftime('%A %d, %B %Y')))
- Что напечатает? И в каком виде?

Преобразование строки в дату – strptime()

- # Воспользуемся библиотекой datetime методом strptime

  from datetime **import** datetime

  print(datetime.strptime('22 04 2020 19:33', '%d %m %Y %H:%M'))
- 2020-04-22 19:33:00

Преобразование даты и времени в строки – strftime()

- from datetime import date, time

- my_date = date(2021, 8, 10)
- my_time = time(7, 18, 34)

- print(my_date) # вывод в ISO формате
- print(my_time) # вывод в ISO формате

- print(my_date.strftime('%d/%m/%y')) # форматированный вывод даты
- print(my_date.strftime('%A %d, %B %Y')) # форматированный вывод даты
- print(my_time.strftime('%H.%M.%S')) # форматированный вывод

Локализация

```
from datetime import date
import locale

locale.setlocale(locale.LC_ALL, "ru")  # иногда используется 'ru_RU.UTF-8’

my_date = date(2021, 8, 10)
print(my_date.strftime("%A %d, %B %Y"))
```

import calendar

- Функция **weekday(year, month, day)** возвращает день недели в виде целого числа (по умолчанию 0 – понедельник, 6 – воскресенье) для заданной даты.
- Функция **monthrange(year, month)** возвращает день недели первого дня месяца и количество дней в месяце в виде кортежа для указанного года year и месяца month
- Функция monthcalendar(year, month) возвращает матрицу, представляющую календарь на месяц. Каждая строка матрицы представляет неделю.
- Функция month(year, month, w=0, l=0) возвращает календарь на месяц в многострочной строке. Аргументами функции являются: year (год), month (месяц), w (ширина столбца даты) и l (количество строк, отводимые на неделю).
- Функция calendar(year, w=2, l=1, c=6, m=3) возвращает календарь на весь год в виде многострочной строки. Аргументами функции являются: year (год), w (ширина столбца даты) и l (количество строк, отводимые на неделю), c (количество пробелов между столбцом месяца), m (количество столбцов).
- Функция isleap(year) определяет, является ли год високосным.

Календарь месяца

```
import calendar
year, month = tuple(map(int, input().split()))
 print(calendar.monthcalendar(year, month))
```

Выполните эту программу, что будет напечатано?

Тернарный оператор

- x = 1
- y = 2
- maximum = x if **x > y** else y

Например, можно так, но очень громоздко:

```
def abs(number):
    if number >= 0:
        return number
    return –number
```

А можно так, более лаконично:

```
def abs(number):
    return number if number >= 0 else -number
```

Задание

- Напишите функцию flip_flop,
- которая возвращает flip, если аргумент равен flop,
- иначе возвращает flop.

- Используйте тернарный оператор
- (что-то вроде return number if number >= 0 else –number)

Списковое включение – list comprehension

- Списковые включения – это способ построения нового списка за счет применения выражения к каждому элементу последовательности.

- Используется с циклом **for,** а также инструкции **if-else** для определения того, что в итоге окажется в финальном списке.

- Иногда не совсем точно называется “генератором” списка, отчего может возникнуть путаница, потому что есть еще генераторные выражения и генераторные функции, которые являются объектами другого типа.

Пример

Do this	For this collection	In this situation
`[x**2`	`for x in range(0, 50)`	`if x % 3 == 0]`

Способы формирования списков

1) при помощи циклов
2) при помощи функции map()
3) при помощи list comprehension

1. При помощи цикла **for**

- s = []
- **for** i in range(10):
- s.append(i ** 3) # Добавляем к списку куб каждого числа
- print(s)
- # [0, 1, 8, 27, 64, 125, 216, 343, 512, 729]

2. При помощи функции **map()**

- list(**map**(lambda x: x ** 3, range(0,10)))
- # [0, 1, 8, 27, 64, 125, 216, 343, 512, 729]

3. При помощи конструкции **list comprehension**

- [x**3 for x in range(10)]
- # [0, 1, 8, 27, 64, 125, 216, 343, 512, 729]

Постройте список из степени 4 всеми тремя способами

(С помощью функции time.time() оцените скорость построения списка)

Условие в конце включения

- **[**«значение» for «элемент списка» in «список» **if** «условие»**]**

- #Получить все нечетные цифры в диапазоне от 0 до 9
- [x for x in range(10) **if** x%2 == 1]
- #[1, 3, 5, 7, 9]

- Постройте список из четных чисел до 20

Условие в начале включения

- # Замена отрицального диапазона нулем
- >>> original_prices = [1.25, -9.45, 10.22, 3.78, -5.92, 1.16]
- >>> prices = [i if i > 0 else 0 **for** i in original_prices]
- >>> prices
- [1.25, 0, 10.22, 3.78, 0, 1.16]

Условие в начале включения

```
from string import ascii_letters
letters = 'hытфтrцзq'  # набор букв из разных алфавитов

# Разграничиваем буквы на английские и не английские
is_eng = [f’{ltr}-ДА’  if  ltr in ascii_letters  else  f’{ltr}-НЕТ’  for ltr in letters]

# ['h-ДА', 'ы-НЕТ', 't-ДА', 'ф-НЕТ', 'т-НЕТ', 'r-ДА', 'ц-НЕТ', 'з-НЕТ', 'q-ДА']
```

Вызов функции в выражении

```
# Замена отрицального числа нулем
original_prices = [1.25, -9.45, 10.22, 3.78, -5.92, 1.16]

def get_price(price):
    return price if price > 0 else 0

prices = [get_price(i) for i in original_prices]

# Перепишите с помощью лямбда функции
```

Вложенные if else

- Постройте функцию FizzBuzz с помощью вложенных if else, функции, списковых включений (делится на 15-> FB, на 3->F, на 5->B)

```
def fu(x):
  return 'FizzBuzz' if x % 15 == 0 else 'Fizz' if x % 3 == 0 else 'Buzz' if x % 5 == 0 else x


a = [fu(x) for x in range(20)]
print(*a)
```

Вложенная генерация

- # Представим список из слов, который мы хотим привести к сплошному списку из букв. Двойная итерация: по словам и по буквам

- **words** = ['Я', 'изучаю', 'Python']

- res = [letter **for** word in words **for** letter in **word**]
- print(res)

- >>>res
- ['Я','и','з','у','ч','а','ю','P','y','t','h','o','n']

Вложенная генерация

- key = ["name", "age", "weight"]
- value = ["Lilu", 25, 100]

- [**{**x, y**}** **for** x **in** key **for** y **in** value] # скобки {} показывают, что создается список, состоящий из **множеств**

- [
- {'Lilu', 'name'}, {25, 'name'}, {100, 'name'}, {'Lilu', 'age'}, {25, 'age'}, {100, 'age’}, {'weight', 'Lilu'}, {'weight', 25}, {'weight', 100}
-]

Вложенная генерация

- >>> matrix = [**[**i for i in range(5)**]** for j in range(6)]
- >>> matrix
- [
- [0, 1, 2, 3, 4],
- [0, 1, 2, 3, 4],
- [0, 1, 2, 3, 4],
- [0, 1, 2, 3, 4],
- [0, 1, 2, 3, 4],
- [0, 1, 2, 3, 4]
-]
- Внешний генератор [... for _ in range(6)] создает 6 строк в то время как внутренний генератор[i for i in range(5)]заполняет каждую строку значениями.
- Поставьте в качестве результата вместо i + j. Что получится? i * j? I ** j?

Вложенная генерация

- # Преобразование матрицы в плоский вид
- matrix = [
- ... [0, 0, 0],
- ... [1, 1, 1],
- ... [2, 2, 2],
- ...]
- >>> flat = [num for row in matrix for num in row]
- >>> flat
- [0, 0, 0, 1, 1, 1, 2, 2, 2]

Вложенная генерация

- # Генерация таблицы умножения от 1 до 5

- t =[[x*y for x in range(1, 6)] for y in range(1, 6)]

- print(t)
- [[1, 2, 3, 4, 5],
- [2, 4, 6, 8, 10],
- [3, 6, 9, 12, 15],
- [4, 8, 12, 16, 20],
- [5, 10, 15, 20, 25]]

Когда использовать списки включений?

- Использовать для выполнения простых фильтраций, модификаций или форматирования итерируемых объектов.
- Для увеличение производительности.
- Для компактности
- Следует избегать использования “генератора” списков, если вам нужно добавить слишком много условий - это делает код трудным для чтения.

Генераторные выражения

- Списковое включение:

  a = [x for x in range(10000)] # Скобки квадратные, все объекты генерируются сразу

- Генераторное выражение:

  b = (x for x in range(10000)) # Скобки круглые, это не кортеж, объекты генерируются по запросу

```
print(type(a), type(b)) # типы создаваемых объектов
import sys
print(sys.getsizeof(a), sys.getsizeof(b))  # размеры объектов
```

Какое число больше? Добавьте пару нулей, как изменились числа?

“Генераторы” словарей – dictionary comprehension

- Генерация словаря похожа на генерацию списка (list comprehension) и предназначена для создания словаря.
- d = {}
- for num in range(1, 10):
- d[num] = num**2
- print(d)
- {1:1, 2:4, 3:9, 4:16, 5:25, 6:36, 7:49, 8:64, 9: 81}
- d = **{** num**2:num for num in range(1, 10)**}**
- >>>d
- {1:1, 2:4, 3:9, 4:16, 5:25, 6:36, 7:49, 8:64, 9: 81}

Генераторы словарей

- #Создадим словарь по списку кортежей
- items = [('c', 3), ('d', 4), ('a', 1), ('b', 2)]
- dict_variable = { **key:value** for (key,value) in items }
- print(dict_variable)
- Что если убрать эту часть строки :value ?
- **Set comprehensions!**

Условие if

- # Добавим в конструкцию генератора условие фильтрации

- dict1 = {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
- # Проверка, больше ли элемент, чем 2

- filtered = {k:v for (k,v) in dict1.items() if v>2}

- print(filtered)
- # {'e': 5, 'c': 3, 'd': 4}

Условие if

- Фильтрация по возрасту

- ages = {
- 'kevin': 12,
- 'marcus': 9,
- 'evan': 31,
- 'nik': 31
- }
- f = {k:val for (k, val) in ages.items() if val > 25}
- print(new_ages)

Несколько условий if

- #Последовательные операторы if работают так, как если бы между ними были логические **and**.

- dict = {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
- r = {k:v for (k,v) in dict.items() if v>2 if v%2 == 0}
- print(r)
- # {'d': 4}

Использование функции enumerate

- names = ['Harry', 'Hermione', 'Ron', 'Neville', 'Luna']
- index = {k:v for (k, v) in enumerate(names)}
- print(index)
- {'Harry':0, 'Hermione':1, 'Ron':2, 'Neville':3, 'Luna’:4}

- Создайте словарь FizzBuzz

Когда использовать генераторы словарей?

- Во всех случаях что и при генерации списков:

- Использовать для выполнения простых фильтраций, модификаций или форматирования итерируемых объектов.

- Для увеличение производительности.

- Для компактности

- Следует избегать использования генератора списков, если вам нужно добавить слишком много условий - это делает код трудным для чтения.

Резюме

- Подобные конструкции позволяют создавать не только списки (list comprehension) и словари (dictionary comprehension), генераторы (generator expression – при помощи «()»), а также множества (set comprehension – при помощи «{}» и кортежи «tuple()»).
- Принцип везде один и тот же.

Задание

- Создайте функцию, которая принимает два аргумента, год и месяц, и возвращает list comprehension, содержащий все даты этого месяца в этом году.
- Используйте функцию **monthrange(year, month)** из модуля calendar для нахождения числа дней в месяце.