УДК 004.6
ББК **32.972**
Ф63

Фислер К., Кришнамурти Ш., Лернер Б. С., Политц Дж. Г.
Ф63 Введение в программирование и структуры данных / пер. с англ. А. В. Снастина. – М.: ДМК Пресс, 2022. – 440 с.: ил.

ISBN 978-5-93700-137-5

В этой книге представлены полезные методики программирования, имеющие практическую ценность. Опираясь на свой многолетний опыт, авторы показывают, как написать надежный код, который смогут читать другие разработчики. Основной принцип обучения -- составление плана решения: от определения структур данных по условиям поставленной задачи через примеры и тесты к написанию программного кода. Обсуждаются типичные ошибки программистов. Наряду с техническими аспектами рассматривается социальное воздействие программ, их выгоды и вред.

Приведенные упражнения позволят читателям самостоятельно закрепить материал.

Книга будет полезна студентам вузов, где преподается информатика, а также тем, кто хочет изучить программирование на базовом уровне.

УДК 004.6
ББК 32.972

ISBN 978-5-93700-137-5 (рус.)

Содержание

От издательства 14

Часть I. ВВЕДЕНИЕ 15

Глава 1. Предисловие 16

1.1. О чем эта книга 16
1.2. Основополагающие принципы, определяющие содержание этой книги 16
1.3. Наш взгляд на данные 17
1.4. Что делает эту книгу особенной 18
1.5. Для кого предназначена эта книга 19
1.6. Структура книги 19
1.7. Организация материала книги 20
1.8. Наш выбор языка программирования 22
1.9. Обратная связь, сообщения об ошибках и комментарии 23

Глава 2. Благодарности 24

Часть II. ОСНОВЫ 26

Глава 3. Начинаем работу 27

3.1. Поясняющий пример: флаги 27
3.2. Числа 28
3.3. Выражения 30
3.4. Терминология 31
3.5. Строки 31
3.6. Изображения 32
    3.6.1. Объединение изображений 33
    3.6.2. Создание флага 34
3.7. Небольшое отступление: типы, ошибки и документация 35
    3.7.1. Типы и контракты 35
    3.7.2. Ошибки формата и нотации 37
    3.7.3. Поиск других функций: документация 38

Глава 4. Именование значений 40

4.1. Панель определений 40
4.2. Именование значений 40
    4.2.1. Сравнение имен и строк 41
    4.2.2. Сравнение выражений с инструкциями 42
    4.3. Каталог программы 43
    4.3.1. Объяснение смысла кнопки Run 44
4.4. Использование имен для оптимизации создания изображений 46

Глава 5. От повторяющихся выражений к функциям 48
5.1. Учебный пример: похожие флаги 48
5.2. Определение функций 49
5.2.1. Как вычисляются функции 50
5.2.2. Аннотации типов 51
5.2.3. Документация 53
5.3. Практическая методика разработки функций: расчет веса на Луне 54
5.4. Документирование функций с примерами 55
5.5. Практическая методика разработки функций: стоимость авторучек 56
5.6. Резюме: определение функций 59

Глава 6. Условные и логические выражения 61
6.1. Учебный пример: вычисление стоимости доставки 61
6.2. Условные выражения: вычисления с принятием решений 62
6.3. Логические выражения 63
6.3.1. Другие логические операции 64
6.3.2. Объединение логических выражений 66
6.4. Как задать сразу несколько вопросов 66
6.5. Вычисление методом упрощения выражений 70
6.6. Совместное использование функций 71
6.6.1. Как вычисляются совместно используемые функции 72
6.6.2. Совместное использование функций и внутренний каталог 73
6.7. Вложенные условные выражения 74
6.8. Резюме: логические и условные выражения 78

Глава 7. Введение в табличные данные 80
7.1. Создание табличных данных 82
7.2. Извлечение значений строк и ячеек 84
7.3. Функции для работы со строками 86
7.4. Обработка строк таблицы 87
7.4.1. Поиск строк 88
7.4.2. Упорядочение строк 89
7.4.3. Добавление новых столбцов 91
7.4.4. Вычисление новых значений столбца 93
7.5. Примеры функций для создания таблиц 94

Глава 8. Обработка таблиц 96
8.1. Очистка таблиц данных 97
8.1.1. Загрузка таблиц данных 97
8.1.2. Обработка отсутствующих элементов 98
8.1.3. Нормализация данных 100
8.1.4. Нормализация, систематическое применение 104
8.1.4.1. Использование программ для обнаружения ошибок в данных 105
8.2. Планирование задач 106
8.3. Подготовка таблиц данных 109
8.3.1. Создание групп по категориям 110

8.3.2. Разделение столбцов ... 110
8.4. Управление и именование таблиц данных ... 112
8.5. Визуальные представления и графики ... 113
8.6. Резюме: управление анализом данных ... 115

Глава 9. От таблиц к спискам ... 117
9.1. Основные статистические вопросы ... 117
9.2. Извлечение столбца из таблицы ... 118
9.3. Объяснение смысла списков ... 119
9.3.1. Списки как анонимные данные ... 119
9.3.2. Создание литеральных списков ... 120
9.4. Операции со списками ... 121
9.4.1. Встроенные операции со списками чисел ... 121
9.4.2. Встроенные операции для любых списков ... 121
9.4.3. Небольшое отступление о соглашениях об именовании ... 122
9.4.4. Получение элементов по позиции ... 123
9.4.5. Преобразование списков ... 124
9.4.6. Резюме: краткий обзор операций для работы со списками ... 125
9.5. Лямбда: анонимные функции ... 127
9.6. Совместное использование списков и таблиц ... 128

Глава 10. Обработка списков ... 131
10.1. Создание списков и разделение их на части ... 131
10.2. Несколько упражнений с примерами ... 134
10.3. Структурированные задачи со скалярными ответами ... 134
10.3.1. `my-len`: примеры ... 134
10.3.2. `my-sum`: примеры ... 136
10.3.3. От примеров к исходному коду ... 137
10.4. Структурированные задачи, в которых выполняется преобразование списков ... 139
10.4.1. `my-doubles`: примеры и код ... 140
10.4.2. `my-str-len`: примеры и код ... 142
10.5. Структурированные задачи, которые выбирают элементы из списков ... 143
10.5.1. `my-pos-nums`: примеры и код ... 143
10.5.2. `my-alternating`: примеры и код ... 145
10.6. Структурированные задачи на нестрогих областях определения ... 148
10.6.1. `my-max`: примеры ... 148
10.6.2. `my-max`: от примеров к коду ... 150
10.7. Более структурированные задачи со скалярными ответами ... 152
10.7.1. `my-avg`: примеры ... 152
10.8. Структурированные задачи с аккумуляторами ... 154
10.8.1. `my-running-sum`: первая попытка ... 154
10.8.2. `my-running-sum`: примеры и код ... 154
10.8.3. `my-alternating`: примеры и код ... 156
10.9. Работа с несколькими ответами ... 157
10.9.1. `uniq`: постановка задачи ... 157

10.9.2. uniq: примеры 157
10.9.3. uniq: код 158
10.9.4. uniq: сокращенное вычисление 160
10.9.5. uniq: пример и варианты кода 161
10.9.6. uniq: почему создается список? 162
10.10. Мономорфные списки и полиморфные типы 162

Глава 11. Введение в структурированные данные 164
11.1. Объяснение типов сложных составных данных 164
11.1.1. Первый взгляд на структурированные данные 164
11.1.2. Первый взгляд на условные данные 165
11.2. Определение и создание структурированных и условных данных 166
11.2.1. Определение и создание структурированных данных 166
11.2.2. Аннотации для структурированных данных 167
11.2.3. Определение и создание условных данных 168
11.3. Программирование со структурированными и условными данными ... 169
11.3.1. Извлечение полей из структурированных данных 169
11.3.2. Различение вариантов условных данных 170
11.3.3. Обработка полей вариантов 171

Глава 12. Наборы структурированных данных 173
12.1. Списки как наборы данных 174
12.2. Множества как наборы данных 176
12.2.1. Выбор элементов из множеств 177
12.2.2. Вычисления с использованием множеств 178
12.3. Сочетание структурированных и объединенных в набор данных 179
12.4. Задача проектирования данных: представление опросов 180

Глава 13. Рекурсивные данные 183
13.1. Функции для обработки рекурсивных данных 185
13.2. Шаблон для обработки рекурсивных данных 190

Глава 14. Деревья 193
14.1. Задача проектирования данных – данные родословной 193
14.1.1. Вычисление генетических родителей по таблице родословной 194
14.1.2. Вычисление прародителей по таблице родословной 196
14.1.3. Создание типа данных для деревьев родословной 197
14.2. Программы для обработки деревьев родословной 199
14.3. Резюме: методика решения задач о деревьях 201
14.4. Учебные вопросы 201

Глава 15. Функции как данные 203
15.1. Немного математического анализа 203
15.2. Удобная сокращенная форма записи для анонимных функций 206
15.3. Потоки из функций 206
15.4. Объединение сил: потоки производных 212

Глава 16. Интерактивные игры как системы с обратной связью 214
16.1. Немного об анимации с обратной связью 215
16.2. Предварительные условия 216
16.3. Версия: самолет пересекает экран 216
16.3.1. Обновление состояния окружающей среды 217
16.3.2. Вывод представления состояния окружающей среды 218
16.3.3. Наблюдение за временем (и совмещение всех элементов) 219
16.4. Версия: непрерывное циклическое движение 220
16.5. Версия: снижение 221
16.5.1. Движение самолета 222
16.5.2. Визуализация сцены 223
16.5.3. Завершающие штрихи 224
16.6. Версия: ответная реакция на нажатия клавиш 224
16.7. Версия: посадка 226
16.8. Версия: закрепленный воздушный шар 228
16.9. Версия: следите за топливным баком 230
16.10. Версия: воздушный шар тоже двигается 232
16.11. Версия: один, два, ... девяносто девять летающих воздушных шаров 233

Глава 17. Примеры, тестирование и проверка программ 234
17.1. От примеров к тестам 234
17.2. Улучшенные сравнения 236
17.3. Когда тесты не проходят 239
17.4. Прогнозирование тестирования 240

Часть III. АЛГОРИТМЫ 242

Глава 18. Прогнозирование роста 243
18.1. Маленькая (правдивая) история 243
18.2. Основной аналитический принцип 247
18.3. Модель стоимости для времени выполнения Pyret 248
18.4. Размер входных данных 249
18.5. Табличный метод для отдельных структурированных рекурсивных функций 250
18.6. Создание рекуррентных последовательностей 252
18.7. Форма записи для функций 254
18.8. Сравнение функций 254
18.9. Объединение O-больших без проблем 256
18.10. Решение рекуррентных последовательностей 257

Глава 19. Обратимся к множествам 261
19.1. Представление множеств с помощью списков 262
19.1.1. Варианты выбора представления 262
19.1.2. Временна́я сложность 263

19.1.3. Выбор одного из представлений 264
19.1.4. Другие операции 266
19.2. Как заставить множества расти на деревьях 267
19.2.1. Преобразование значений в упорядоченные значения 268
19.2.2. Использование двоичных деревьев 270
19.2.3. Точный баланс: обрезка деревьев 274
19.2.3.1. Вариант левое–левое 277
19.2.3.2. Вариант левое–правое 278
19.2.3.3. Существуют ли какие-либо другие варианты? 279

Глава 20. Хэллоуин-анализ 281
20.1. Первый пример 281
20.2. Новая форма анализа 281
20.3. Пример: очереди из списков 282
20.3.1. Представления в виде списка 282
20.3.2. Первоначальный анализ 283
20.3.3. Более разнообразные последовательности операций 283
20.3.4. Второй этап анализа 285
20.3.5. Сравнение амортизации с отдельными операциями 285
20.4. Материал для дополнительного чтения 285

Глава 21. Совместное использование значений и равенство 286
21.1. Новый взгляд на равенство 286
21.2. Стоимость вычисления ссылок 290
21.3. Формы записи равенства 292
21.4. В интернете никто не знает, что вы НАГ 292
21.5. НАГ был всегда 294
21.6. От ацикличности к циклам 295

Глава 22. Графы 297
22.1. Объяснение сущности графов 297
22.2. Представления 301
22.2.1. Связи по имени 301
22.2.2. Связи по индексам 303
22.2.3. Список ребер 305
22.2.4. Абстрагирующие представления 306
22.3. Измерение сложности для графов 306
22.4. Достижимость 307
22.4.1. Простая рекурсия 307
22.4.2. Приведение в порядок цикла 308
22.4.3. Проход с использованием памяти 309
22.4.4. Улучшенный интерфейс 310
22.5. Обход в глубину и в ширину 311
22.6. Графы со взвешенными ребрами 312
22.7. Наикратчайшие (или наилегчайшие) пути 313
22.8. Моравские остовные деревья 315
22.8.1. Глобальная задача 316

22.8.2. Жадное решение ... 316
22.8.3. Другое жадное решение ... 317
22.8.4. Третье решение ... 318
22.8.5. Проверка связности компонентов ... 319

Часть IV. ОТ PYRET К PYTHON ... 324

Глава 23. От Pyret к Python ... 325

23.1. Выражения, функции и типы ... 325
23.2. Возврат значений из функций ... 327
23.3. Примеры и варианты тестов ... 328
23.4. Небольшое отступление по поводу чисел ... 329
23.5. Условные выражения ... 331
23.6. Создание и обработка списков ... 331
23.6.1. Фильтры, отображения и друзья ... 332
23.7. Данные с компонентами ... 333
23.7.1. Доступ к полям внутри классов данных ... 334
23.8. Обход списков ... 334
23.8.1. Представляем циклы `for` ... 334
23.8.1.1. Небольшое отступление о порядке обработки элементов списка ... 336
23.8.2. Использование циклов `for` в функциях, создающих списки ... 337
23.8.3. Резюме: шаблон обработки списков для Python ... 338

Часть V. ПРОГРАММИРОВАНИЕ С СОХРАНЕНИЕМ СОСТОЯНИЯ ... 339

Глава 24. Изменение структурированных данных ... 340

24.1. Изменение полей структурированных данных ... 341
24.2. Изменение совместно используемых данных ... 342
24.3. Объяснение функционирования памяти ... 344
24.4. Переменные и равенство ... 345
24.5. Хранение простых данных в памяти ... 346

Глава 25. Изменение переменных ... 348

25.1. Изменение переменных в памяти ... 348
25.2. Изменение переменных, связанных со списками ... 352
25.3. Создание функций, изменяющих переменные ... 353
25.3.1. Аннотация `global` ... 354
25.4. Тестирование функций, изменяющих глобальные переменные ... 355
25.4.1. Внутренняя структура функции тестирования ... 359
25.4.2. Общие выводы о тестировании изменений ... 359

Глава 26. Возврат к спискам и переменным ... 361

26.1. Обновление совместно используемого списка ... 361
26.1.1. Операции, изменяющие списки ... 362
26.2. Списки в памяти ... 363

26.3. Практический пример: данные для совместно используемых банковских счетов ... 365
26.4. Циклические ссылки ... 369
26.4.1. Тестирование циклических данных ... 371
26.4.2. Снова переменные: функция для создания счетов для новых клиентов ... 371
26.5. Многочисленные роли переменных ... 372
26.6. Управление всеми счетами ... 373

Глава 27. Хеш-таблицы и словари ... 375

27.1. Поиск по условиям, отличающимся от ключей ... 376
27.2. Словари с более сложными значениями ... 377
27.3. Использование структурированных данных в качестве ключей ... 378

Часть VI. ДОПОЛНИТЕЛЬНЫЕ ТЕМЫ ... 380

Глава 28. Алгоритмы, использующие состояние ... 381

28.1. И снова о непересекающихся множествах ... 381
28.1.1. Оптимизации ... 382
28.1.2. Анализ ... 383
28.2. Установка членства методом обратного хеширования ... 384
28.2.1. Улучшение времени доступа ... 385
28.2.2. Улучшенное хеширование ... 387
28.2.3. Фильтры Блума ... 387
28.3. Устранение повторных вычислений с помощью запоминания ответов ... 389
28.3.1. Интересная числовая последовательность ... 389
28.3.1.1. Использование состояния для запоминания предыдущих ответов ... 391
28.3.1.2. От дерева вычислений к НАГ ... 392
28.3.1.3. Сложность чисел ... 393
28.3.1.4. Абстрагирование мемоизации ... 393
28.3.2. Редакторское расстояние для исправления орфографических ошибок ... 395
28.3.3. Природа как машинистка с неловкими пальцами ... 400
28.3.4. Динамическое программирование ... 401
28.3.4.1. Числа Каталана и динамическое программирование ... 402
28.3.4.2. Расстояние Левенштейна и динамическое программирование ... 403
28.3.5. Сравнение мемоизации и динамического программирования ... 406

Часть VII. ПРИЛОЖЕНИЯ ... 409

Глава 29. Pyret для пользователей Racket и Scheme ... 410

29.1. Числа, строки и логические значения ... 410
29.2. Инфиксные выражения ... 411
29.3. Определение и применение функций ... 411

29.4. Тесты 412
29.5. Имена переменных 413
29.6. Определения данных 413
29.7. Условные выражения 415
29.8. Списки 417
29.9. Функции первого класса 418
29.10. Аннотации 418
29.11. Что еще? 418

Глава 30. Сравнение Pyret с Python 419

Глава 31. Сравнение этой книги с HtDP 422

Глава 32. Примечания к текущей редакции книги 425

Глава 33. Словарь терминов 426

Предметный указатель 429

Часть I

ВВЕДЕНИЕ

Глава 1

Предисловие

1.1. О чем эта книга

Эта книга представляет собой введение в информатику. Она научит вас программировать такими способами, которые имеют практическую ценность и важность. Но этот процесс обучения будет к тому же выходить за рамки программирования в более широкую область информатики, богатой, глубокой, увлекательной и прекрасной многогранной дисциплины. Вы узнаете много полезных вещей, которые сможете сразу же применить на практике, но мы также продемонстрируем вам кое-что из того, что скрывается за ними.

Прежде всего мы хотим предоставить вам способы мышления для решения задач с помощью вычислений. Некоторые из этих способов являются техническими методиками, такими как обработка данных и примеры для разработки решений задач. Другие представляют собой научные методы, такие как способы, позволяющие убедиться в том, что программы надежны и делают именно то, для чего они предназначены. Наконец, некоторые из таких способов имеют социальную составляющую, они заставляют задуматься о влиянии программ на людей.

1.2. Основополагающие принципы, определяющие содержание этой книги

Наша точка зрения основана на многолетнем опыте работы в качестве разработчиков программного обеспечения, исследователей и преподавателей. Эти виды деятельности позволили нам выработать следующие твердые убеждения:

- программное обеспечение пишется не только для того, чтобы его выполнять. Программы следует писать так, чтобы их могли читать и сопровождать другие люди. Часто этим «другим» человеком являетесь вы сами, через шесть месяцев забывшие, что было сделано и почему именно так;

- программисты несут ответственность за то, чтобы созданное ими программное обеспечение соответствовало поставленным целям и являлось надежным. Это требование отражено в различных дисциплинах информатики, таких как тестирование и проверка (верификация) программ;
- программы должны быть предсказуемыми. Мы должны знать, насколько это возможно, как будет вести себя программа, до ее запуска. Это поведение включает в себя не только технические характеристики, такие как время работы, использование пространства памяти, мощность и т. п., но и социальные воздействия и последствия, выгоды и вред. Программисты, как известно, далеко не всегда задумываются о последнем.

1.3. Наш взгляд на данные

Высказанные выше опасения пересекаются с нашими представлениями о развитии информатики как отдельной дисциплины. Все прекрасно понимают, что мы живем в мире, переполненном данными, но к каким последствиям это приводит?

На вычислительном уровне влияние данных стало огромным. Обычно единственным способом сделать программу лучше было непосредственное ее усовершенствование, что часто означало усложнение и воздействие на факторы, которые мы обсуждали выше. Но есть категории программ, для которых существует другой метод: просто передать в ту же программу больше данных или данные лучшего качества, и программа, возможно, улучшится. Эти управляемые данными программы лежат в основе многих инноваций, которые мы наблюдаем в окружающем мире.

В дополнение к этому техническому влиянию данные могут иметь еще и глубокое педагогическое воздействие. В большинстве случаев процессу начального обучения программированию наносят ущерб придуманные данные, которые не имеют реального смысла, не вызывают никакого интереса и не дают никаких практических результатов (а также часто сопровождаются искусственно созданными проблемами). Имея в своем распоряжении реальные данные, обучающиеся могут самостоятельно регулировать процесс обучения, сосредоточившись на задачах, которые они считают практически целесообразными, более информативными или просто интересными, задавая вопросы и отвечая на те из них, которые они считают наиболее полезными. Разумеется, с этой точки зрения программы запрашивают данные: т. е. программы являются инструментами для ответов на вопросы. В свою очередь, особое внимание, уделяемое реальным данным и ответам на практические вопросы, позволяет нам обсуждать социальное воздействие информатики и вычислительной техники.

Эти явления породили совершенно новые области исследований, обычно называемые наукой о данных. Но типовые учебные программы по науке о данных также имеют много ограничений. Они уделяют слишком мало

внимания тому, что нам известно о трудностях обучения программированию, а также надежности программного обеспечения. И такие учебные программы абсолютно не учитывают того, что их данные часто весьма ограничены по своей структуре. На этих ограничениях обычно заканчивается наука о данных и начинается информатика. В частности, структура данных служит отправным пунктом для размышлений и реализации некоторых из вышеперечисленных основополагающих принципов – производительности, надежности и предсказуемости – с использованием многочисленных инструментальных средств информатики.

1.4. Что делает эту книгу особенной

Во-первых, мы предлагаем новую точку зрения на структурирование учебных программ по информатике, которую мы называем ориентацией на данные. Мы рассматриваем учебную программу, ориентированную на данные, как

Более подробно об этом подходе читайте в нашем эссе (https://cs.brown.edu/~sk/Publications/Papers/Published/kf-data-centric/).

ориентация на данные = наука о данных + структуры данных

именно в этом порядке: начинаем с основных принципов науки о данных, затем переходим к классическим принципам для структур данных и прочих разделов информатики. Эта книга излагает такую точку зрения конкретно и подробно.

Во-вторых, в компьютерном образовании много говорится о гипотетических машинах – абстракциях поведения программ, предназначенных для того, чтобы помочь учащимся понять, как эти программы работают, – но в действительности они используются лишь в немногих учебных программах. Мы серьезно относимся к гипотетическим машинам, разрабатывая последовательный ряд таких машин и вводя их в учебную программу. Это связано с нашим убеждением о том, что программы – это не только объекты, которые выполняются, но и объекты, которые мы подробно рассматриваем.

В-третьих, мы включаем в текст книги контент о социальной ответственности информатики. В отличие от других усилий, направленных на ознакомление обучающихся с этикой или со скрытыми потенциальными опасностями технологии в целом, мы стремимся показать обучающимся, как конструкции и концепции, которые они прямо сейчас превращают в код, могут привести к неблагоприятным последствиям, если использовать их, пренебрегая осторожностью. Сохраняя сосредоточенность на тестировании и конкретных примерах, мы вводим несколько тем, заставляющих обучающихся принимать во внимание предположения на уровне конкретных данных. Этот материал приводится в явной форме на протяжении всей книги.

Наконец, эта книга основана на результатах недавних, постоянно продолжающихся научных исследований. Выбор учебного материала, порядок его представления, методы программирования и многое другое основаны на том, что мы знаем из научно-исследовательской литературы. Во многих слу-

чаях мы сами проводим исследования, поэтому учебная программа и научные исследования образуют своеобразный симбиоз. Вы можете найти наши материалы (некоторые написаны совместно, некоторые – в сотрудничестве с другими авторами) на соответствующих страницах (https://www.ccs.neu.edu/home/blerner/papers.html).

1.5. Для кого предназначена эта книга

Эта книга написана в основном для студентов начальных курсов, изучающих информатику в высшем учебном заведении (колледж или университет). Однако многие – особенно начальные – части этой книги также подходят для среднего образования (например, в США приблизительно для 6–12 классов, или для возраста 12–18 лет). В действительности мы видим естественную преемственность между средним и высшим образованием и считаем, что эта книга может послужить полезным связующим звеном между ними.

1.6. Структура книги

В отличие от некоторых других учебных пособий, эта книга не придерживается строго иерархического порядка изложения материала. Скорее, это непрерывно продолжающееся обсуждение с возвратами к предыдущим темам. Чаще всего мы будем создавать программы постепенно, как это могла бы делать пара сотрудничающих программистов. Мы будем совершать ошибки, но не потому, что не знаем, как сделать все правильно, а потому, что для вас это самый лучший способ обучения. Наличие ошибок делает невозможным пассивное чтение: вы непременно должны глубоко осваивать материал, потому что нет никакой уверенности в достоверности предлагаемого для чтения текста.

В итоге вы всегда будете получать правильный ответ. Однако такой нелинейный путь в краткосрочной перспективе вызывает больше разочарований (у вас часто будет возникать соблазн сказать: «Просто сразу скажите мне правильный ответ!») и делает книгу плохим справочным пособием (невозможно открыть случайную страницу с уверенностью в том, что все написанное на ней правильно). Но такое чувство разочарования является правильным восприятием обучения. Другого пути мы не знаем.

Мы используем визуальное форматирование, чтобы выделить некоторые из этих моментов. Таким образом, время от времени вы будете встречать следующие выделенные фрагменты:

Упражнение

Это упражнение. Попробуйте выполнить его.

Это обычное упражнение в учебной литературе. Его нужно выполнить самостоятельно. Если вы используете эту книгу как часть учебного курса, то такое упражнение вполне можно предложить в качестве домашнего задания. Но кроме обычных упражнений вам также будут встречаться похожие на них вопросы, которые выглядят следующим образом:

> ***Выполните прямо сейчас***
> Здесь выполняется некоторое действие. Вы его видите?

Когда встречается один из таких выделенных фрагментов, остановитесь. Прочтите выделенный текст, подумайте и сформулируйте ответ, прежде чем продолжить чтение. Вы обязательно должны сделать именно так, потому что в действительности это упражнение, но ответ уже есть в книге – чаще всего в тексте, непосредственно следующем за ним (т. е. в той части, которую вы читаете прямо сейчас), – или вы можете найти ответ самостоятельно, выполнив программу. Если вы просто продолжите читать дальше, то получите ответ, не думая самостоятельно (или не увидите его вообще, если инструкции предназначены для запуска программы), но в результате (а) не сможете проверить свои знания и (б) не улучшите свою интуицию. Другими словами, в выделенных фрагментах представлены очевидные дополнительные попытки поощрения активного обучения. Но в конечном счете мы можем только поощрять стремление к активному обучению, однако его практическое применение зависит только от вас.

Специальные стратегии проектирования и разработки программ приводятся в блоках текста, выделенных следующим образом:

> **Стратегия: как это сделать...**
>
> Здесь располагается краткое описание способа сделать что-либо.

Наконец, подобным образом выделяется материал по социально ответственному применению информатики – в виде показанного ниже блока текста:

> **Ответственное применение информатики: вы учитывали, что...**
>
> Здесь размещается описание социальных опасностей, возникающих из-за необдуманного использования материала.

1.7. Организация материала книги

Книга содержит четыре части:

1) часть II – введение в программирование для начинающих, которые обучаются программированию и элементарному анализу данных.

В ней представлены основные концепции программирования посредством создания изображений и обработки таблиц, затем рассматриваются списки, деревья и написание программ, реагирующих на действия пользователя, и все это с учетом ориентации на данные. Условная гипотетическая машина в этом разделе основана на подстановке;

2) часть III – здесь рассматривается асимптотическая сложность, рекуррентные выражения и основные графовые алгоритмы;
3) часть V – рассматривается работа с изменяемыми переменными и изменяемыми структурированными данными, формируется понимание изменяемых списков и хеш-таблиц (и их обработки). В этом разделе мы переходим на язык Python. Он позволяет расширить возможности тестирования, чтобы подробно рассмотреть нюансы программ с динамическими изменениями. Условная гипотетическая машина в этой части отделяет среду именования (называемую здесь каталогом (directory)) от динамически распределяемой памяти (кучи) значений структурированных данных;
4) часть VI – здесь мы возвращаемся к темам алгоритмов, основанных на оценке состояния и на структурах данных с отслеживанием их состояния.

Эти части были тщательно разработаны, чтобы обеспечить отсутствие зависимостей между частями III и V. Это позволяет гибко планировать предложения нескольких различных типов учебных курсов. Например, реорганизовав материал этих частей, мы уже сейчас предлагаем два совершенно различных курса, которые могли бы использовать другие читатели:

- вводный курс может использовать части II и V (без части III) для представления ориентированной на данные точки зрения информатики и обучения студентов основным навыкам программирования на языке Python;
- более продвинутый курс, предполагающий, что обучающиеся уже знакомы с основами функционального программирования (например, по первым частям книги «How to Design Programs» (https://htdp.org/) («Как проектировать программы»), можно было бы начать непосредственно с части III или, возможно, с отдельных разделов части II для включения недостающего материала (например, работы с таблицами). Этот курс может быть продолжен в части V, за которой следует часть VI.

Описанные выше курсы аналогичны соответственно учебным курсам CSCI 0111 (https://cs.brown.edu/courses/csci0111/) и CSCI 0190 (https://cs.brown.edu/courses/csci0190/) в университете Брауна (Провиденс, Род-Айленд, США). На указанных здесь страницах хранятся все предыдущие экземпляры курсов, включая все задания и сопутствующие материалы. Читатели могут использовать их в своих учебных курсах.

Многие из этих курсов будут изучать студенты, которые ранее занимались программированием с сохранением состояния (на Python, Java, Scratch или других языках). По нашему опыту, большинство таких студентов получали либо весьма неполные, либо откровенно вводящие в заблуждение

объяснения и метафоры состояния (например, «переменная – это (черный) ящик»). Из-за этого они плохо понимают излагаемый здесь материал, за исключением самых элементарных основ, особенно когда они начинают изучать такие важные темы, как алиасинг (альтернативные имена переменных). В результате многие из этих студентов сочли одновременно новым и весьма познавательным надлежащее объяснение того, как на самом деле работать с состоянием, с помощью нашей искусственной гипотетической машины. По этой причине мы рекомендуем проходить этот материал медленно и внимательно.

Разумеется, мы предлагаем читателям создавать собственные комбинации из глав, входящих в вышеперечисленные части книги. Мы очень хотели бы узнать о других проектах.

1.8. Наш выбор языка программирования

Если бы мы хотели разбогатеть, то написали бы эту книгу полностью на Python. На момент написания книги Python переживает свой расцвет как язык для учебных целей (точно так же, как Java до него, C++ до этого, ранее C, а еще раньше Pascal и так далее). Вне всякого сомнения, у Python есть много привлекательных свойств, и не в последнюю очередь его присутствие в верхних позициях списков вакансий. Но Python неоднократно разочаровывал нас как отправной пункт для изучения программирования (см. главу 30).

В итоге в нашей книге используются два языка программирования. Сначала применяется язык под названием Pyret (https://www.pyret.org/), который мы специально спроектировали и создали для собственных потребностей, как следствие наших разочарований. Он был специально разработан для стиля программирования, описанного в этой книге, поэтому оба языка могут гармонично сосуществовать. Pyret основан на Python, а также на многих других превосходных языках программирования. Таким образом, начинающие программисты получат именно те знания, которые им необходимы, в то время как программистам, уже хорошо знакомым с языковым зверинцем, от змей до верблюдов, Pyret должен показаться знакомым и удобным.

Далее, учитывая значимость Python как стандартного языка обмена информацией и наличие расширяющих его возможности библиотек, в части V книги Python рассматривается весьма подробно. Вместо того чтобы начинать с нуля в Python, мы предлагаем систематический и постепенный переход к этому языку на основе ранее изученного материала. Мы считаем, что это поможет вам изучить программирование в целом лучше, чем если бы вы были знакомы только с одним языком программирования. Но в то же время мы считаем, что такой подход поможет вам лучше понять Python: подобно тому, как вы начинаете больше ценить свой язык, страну и культуру, когда покидаете ее пределы и знакомитесь с другими странами и культурами.

1.9. Обратная связь, сообщения об ошибках и комментарии

Работая с этой книгой, вы, возможно, обнаружите опечатки, найдете фрагменты, которые могли бы быть изложены более понятным языком, или у вас могут возникнуть предложения по качеству следующего издания. Всю эту информацию вы можете передать нам, заполнив опросную форму на общедоступном сайте GitHub (https://github.com/data-centric-computing/dcic-public). Заранее благодарим за сотрудничество.

Часть II

ОСНОВЫ

Глава 3

Начинаем работу

3.1. Поясняющий пример: флаги

Предположим, что вы создаете компанию, занимающуюся графическим дизайном, и хотите иметь возможность создания изображений флагов разных размеров и конфигураций для своих клиентов. На рис. 3.1 показаны образцы изображений, которые необходимо создать с помощью используемого вами программного обеспечения.

Рис. 3.1 ❖ Образцы флагов

Прежде чем пытаться написать исходный код для создания этих разнообразных изображений, вы должны вернуться на шаг назад, внимательно рассмотреть этот набор изображений и попробовать определить их свойства, которые, возможно, помогут нам принять решение о том, что именно нужно сделать. Чтобы действительно получить такую помощь, ответим на пару конкретных вопросов, способствующих лучшему пониманию смысла предлагаемых изображений:

- на что вы обращаете особое внимание в изображениях флагов?

- какие вопросы возникают у вас по изображениям этих флагов или по программе, которая может создавать эти изображения?

> ***Выполните прямо сейчас***
>
> Настоятельно рекомендуем действительно записать свои ответы на приведенные выше вопросы. Внимательное отношение к свойствам данных и информации – чрезвычайно важный навык в информатике.

Возможно, вы обратили внимание на следующие особенности:

- некоторые флаги имеют одинаковую структуру и отличаются только цветами;
- некоторые флаги имеют различные размеры;
- у некоторых флагов есть флагшток;
- большинство изображений выглядят относительно простыми, но некоторые реальные флаги содержат более сложные изображения

…и т. д.

Возможно, у вас возникли следующие вопросы:

- требуется ли возможность рисования этих изображений вручную?
- существует ли возможность формировать изображения флагов различных размеров с помощью единого исходного кода?
- что, если предлагается изобразить флаг, форма которого отличается от прямоугольной?

…и т. д.

Свойства, на которые мы обратили внимание, предполагают те же условия, которые необходимо выполнить для написания программы генерации изображений флагов:

- возможно, потребуется вычисление высоты полос по общим размерам флага (т. е. мы будем писать программы с использованием чисел (numbers));
- необходим способ описания цветов в нашей программе (мы будем изучать (символьные) строки (strings));
- необходим способ создания изображений на основе простых фигур разнообразных цветов (мы будем создавать и объединять выражения (expressions)).

Давайте начнем работать.

3.2. Числа

Начнем с простого примера: вычисление суммы чисел 3 и 5.

Чтобы выполнить это вычисление с помощью компьютера, необходимо каким-то образом записать это вычисление и предложить компьютеру выполнить (run или evaluate) его так, чтобы мы получили результат в виде числа. Программное обеспечение или веб-приложение, в котором вы пишете и запускаете программы, называется средой программирования (средой

разработки программ – programming environment). В первой части нашего учебного курса мы будем использовать язык программирования под названием Pyret.

Перейдите в онлайновый редактор (https://code.pyret.org/editor) (который здесь и далее мы будем называть CPO). Пока что мы будем работать только в правой части этого редактора (в интерактивной панели (interactions pane)).

Группа символов >>> называется промптом (prompt), или приглашением, – здесь мы сообщаем CPO о необходимости выполнить программу. Предложим выполнить сложение чисел 3 и 5. Вот что мы должны ввести после промпта:

```
>>> 3 + 5
```

Нажмите клавишу **Ввод** (Return или Enter), и результат вычисления появится в строке под промптом, как показано ниже:

```
8
```

Результат вполне ожидаем, и мы можем выполнять другие арифметические вычисления:

```
>>> 2 * 6
12
```

(Обратите внимание: с помощью символа * мы записываем знак умножения.)

А если попробовать вычислить 3 + 4 * 5?

> ***Выполните прямо сейчас***
>
> Попробуйте вычислить приведенное выше выражение. Посмотрите, что на это скажет Pyret.

Pyret выдает сообщение об ошибке (error message). Это говорит о том, что Pyret не уверен в том, что именно мы имели в виду:

```
(3 + 4) * 5
```

или

```
3 + (4 * 5),
```

поэтому предлагает добавить круглые скобки, чтобы смысл выражения стал очевидным. В каждом языке программирования существует набор правил, определяющий способ записи программ. Правила Pyret требуют использования круглых скобок для устранения неоднозначности.

```
>>> (3 + 4) * 5
35
>>> 3 + (4 * 5)
23
```

Другое правило Pyret требует наличия пробелов, окружающих арифметические операторы. Давайте посмотрим, что произойдет, если забыть об этих пробелах:

```
>>> 3+4
```

Pyret выведет другое сообщение об ошибке, в котором выделена (подсвечена) та часть исходного кода, которая отформатирована неправильно, а также описание проблемы, которую обнаружил Pyret. Для исправления этой ошибки можно нажать клавишу со стрелкой вверх, находясь в правой панели редактора, и исправить показанное выше выражение, добавив требуемые пробелы.

Выполните прямо сейчас

Попробуйте выполнить описанные выше действия прямо сейчас и убедитесь в том, что все происходит именно так, как описано.

А что, если необходимо применить что-то, отличающееся от основных арифметических операторов? Скажем, нужно выбрать минимальное из двух чисел. Тогда мы должны записать это следующим образом:

```
>>> num-min(2, 8)
```

3.3. Выражения

Обратите внимание: когда мы выполняем `num-min`, то получаем в ответ число (так же, как при использовании +, *, ...). Это означает потенциальную возможность использования результата вычисления `num-min` в других вычислениях, где ожидается числовое значение:

Почему `num-`? Потому что «определение минимума» – это концепция, которая имеет смысл и для данных, отличающихся от чисел. Pyret называет этот оператор вычисления минимума `num-min`, чтобы устранить неоднозначность.

```
››› 5 * num-min(2, 8)
10
››› (1 + 5) * num-min(2, 8)
12
```

Надеемся, здесь вы начинаете видеть закономерность. Мы можем формировать более сложные вычисления из меньших, элементарных, используя операторы для объединения результатов элементарных вычислений. Мы будем использовать термин «выражение» (expression) для обозначения вычислений, записанных в формате, который Pyret может понять и вычислить правильный результат.

Упражнение 3.1

В редакторе CPO попробуйте ввести выражения для каждого из приведенных ниже вычислений:

- вычесть 3 из 7, затем умножить результат на 4;
- вычесть 3 из произведения 7 и 4;
- сложить 3 и 5, сумму разделить на 2;
- определить максимальное значение из пары 5 – 10 и –20;
- 2 разделить на сумму 3 и 5.

Выполните прямо сейчас

А если в результате вы получаете дробь?

Если вы не вполне уверены в том, как получить дробь, то подскажем: для этого существуют два способа: можно ввести выражение, результатом которого является дробь, или просто напрямую ввести дробь (например, `1/3`).

В любом случае вы можете щелкнуть левой кнопкой мыши по результату в интерактивной панели, чтобы изменить представление числа. Попробуйте прямо сейчас.

3.4. Терминология

В интерактивном режиме рассмотрим вычисление следующего выражения:

```
››› (3 + 4) * (5 + 1)
42
```

В действительности в этом интерактивном взаимодействии содержится несколько типов информации, и мы должны дать имя каждому элементу:

- выражение (expression) – вычисление, записанное в формальной нотации какого-либо языка программирования.
  Примеры выражений: `4`, `5 + 1` и `(3 + 4) * (5 + 1)`;
- значение (value) – выражение, для которого уже невозможно продолжить вычисления (т. е. оно само по себе является результатом).
  До сих пор единственными значениями, которые мы наблюдали, являлись числа;
- программа (program) – последовательность выражений, которые необходимо выполнить (вычислить).

3.5. Строки

А если необходимо написать программу, в которой используется информация, отличающаяся от чисел, например чье-то имя? Для имен и прочих текстовых данных используется то, что называется строками (strings). Ниже приведено несколько примеров:

```
"Kathi"
"Go Bears!"
```

```
"CSCI0111"
"Carberry, Josiah"
```

Что здесь можно заметить? Строки могут содержать пробелы, знаки пунктуации и цифры. Мы используем строки для представления текстовых данных. Для примера с созданием флагов мы будем использовать строки для обозначения цветов: `"red"`, `"blue"` и т. д.

Следует отметить, что в строках учитываются различия в регистрах букв (case-sensitive), т. е. прописные (заглавные) буквы отличаются от строчных (скоро мы увидим, что это означает на практике).

3.6. Изображения

Мы уже познакомились с двумя типами данных: числами и строками. Для флагов также потребуются графические изображения. Изображения отличаются от чисел и строк (невозможно описать все изображение в целом с помощью одного числа – ну, если только после более глубокого освоения информатики, но давайте не будем забегать вперед).

В Pyret встроена поддержка графических изображений. После запуска Pyret вы увидите окрашенную в светло-серый цвет строку `"use context essentials2021"` (или что-то подобное). Эта строка обеспечивает конфигурирование Pyret с некоторой базовой функциональностью, дополняющей поддержку чисел и строк.

> ***Выполните прямо сейчас***
>
> Нажмите кнопку **Run** (для активизации функциональных возможностей контекста `essentials`), затем введите каждое из приведенных ниже выражений Pyret после интерактивного промпта, чтобы увидеть результат их выполнения:
>
> - `circle(30, "solid", "red")`
> - `circle(30, "outline", "blue")`
> - `rectangle(20, 10, "solid", "purple")`

Каждое из этих выражений начинается с имени изображаемой фигуры, затем в круглых скобках определяется ее конфигурация. Информация о конфигурации состоит из размеров (радиуса для окружностей, ширины и высоты для прямоугольников – все размеры заданы в экранных пикселах), строки, определяющей, выводится ли сплошная (закрашенная) фигура или только ее контур, а далее следует строка с названием цвета, используемого для отображения фигуры.

Какие фигуры и цвета известны Pyret? Отложим ответ на этот вопрос на некоторое время. Скоро мы продемонстрируем, как находить такую информацию в документации.

3.6.1. Объединение изображений

Ранее мы уже узнали о возможности использования таких операций, как + и *, для объединения чисел в выражения. Каждый раз, когда встречается новый тип данных в программировании, вы должны поинтересоваться, какие операции языка предоставляются для работы с этим типом. Для изображений в Pyret набор операций предоставляет следующие возможности:

- вращение;
- масштабирование;
- транспонирование;
- размещение двух изображений рядом;
- размещение одного изображения поверх другого
- и другие.

Рассмотрим подробнее, как используются некоторые из этих операций.

Упражнение 3.2

Введите приведенные ниже выражения в онлайновом редакторе Pyret:

```
rotate(45, rectangle(20, 30, "solid", "red"))
```

Что означает число `45`? Попробуйте вводить другие числа вместо `45`, чтобы подтвердить или опровергнуть свое предположение.

```
overlay(circle(25, "solid", "yellow"), rectangle(50, 50, "solid", "blue"))
```

Вы можете описать обычным языком, что делает `overlay`?

```
above(circle(25, "solid", "red"), rectangle(30, 50, "solid", "blue"))
```

Какой тип значения вы получаете при использовании операций `rotate` и `above`? (Подсказка: ответом должен быть один из вариантов: число, строка или изображение.)

Эти примеры позволяют нам немного более глубоко задуматься о выражениях. В нашем распоряжении находятся простые значения, такие как числа и строки. У нас есть операции или функции, которые объединяют значения, например `+` или `rotate` («функции» – это термин, гораздо чаще используемый в информатике, тогда как на уроках математики предпочитают термин «операции»). Каждая функция генерирует значение, которое можно использовать как ввод (входные данные) для другой функции. Мы формируем выражения, используя значения и выходные данные функций как входные данные для других функций.

Например, мы используем `above` для создания изображения из двух изображений меньшего размера. Можно было бы взять полученное изображение и повернуть его, воспользовавшись приведенным ниже выражением:

```
rotate(45,
  above(circle(25, "solid", "red"),
    rectangle(30, 50, "solid", "blue")))
```

Принцип использования вывода одной функции как ввода для другой функции известен под названием «композиция» (composition). Самые интересные программы получаются в результате композиции разнообразных вычислений. Уверенное владение навыком составления выражений – важнейший первый шаг в обучении программированию.

Упражнение 3.3

Попробуйте создать следующие изображения:

- синий треугольник (размер выберите сами). Наряду с `circle` существует функция `triangle`, которая принимает длину стороны, стиль заполнения (закраски) и ее цвет и создает изображение равностороннего треугольника;
- синий треугольник внутри желтого прямоугольника;
- треугольник с поворотом на некоторый угол;
- мишень с тремя вложенными окружностями, выровненными по их центрам (например, как на логотипе Target (https://ru.wikipedia.org/wiki/Target));
- любое изображение по вашему выбору – поэкспериментируйте в свое удовольствие.

Создание изображения мишени может оказаться немного более сложным. Функция `overlay` принимает только два изображения, поэтому вам придется поразмыслить о том, как использовать композицию для правильного размещения трех концентрических окружностей.

3.6.2. Создание флага

Мы полностью готовы к созданию первого флага. Начнем с флага Армении, который содержит три горизонтальные полосы: красная сверху, синяя в середине и оранжевая внизу.

Упражнение 3.4

Используйте ранее изученные функции для создания изображения флага Армении. Выберите размеры самостоятельно (мы рекомендуем ширину от 100 до 300 пикселов).

Запишите список вопросов и идей, которые возникают во время выполнения этого упражнения.

3.7. Небольшое отступление: типы, ошибки и документация

Теперь, когда вы получили полное представление о том, как создается изображение флага, вернемся немного назад и чуть более подробно рассмотрим две концепции, с которыми вы уже встречались: типы и сообщения об ошибках.

3.7.1. Типы и контракты

Теперь, когда мы научились объединять функции для построения более сложных выражений из простых, нам придется тщательно следить за тем, чтобы составляемые комбинации имели смысл. Рассмотрим следующий пример кода Pyret:

```
8 * circle(25, "solid", "red")
```

Как вы думаете, какой результат будет получен при вычислении этого выражения? Умножение имеет смысл при работе с числами, но этот код предлагает Pyret умножить число на изображение. Имеет ли это действие какой-либо смысл?

Этот код не имеет никакого смысла, поэтому Pyret, разумеется, выведет сообщение об ошибке, если вы попытаетесь выполнить его.

> ***Выполните прямо сейчас***
>
> Попробуйте выполнить приведенный выше код, затем рассмотрите выведенное сообщение об ошибке. Запишите информацию, которую это сообщение предоставляет, о том, что пошло не так (скоро мы вернемся к этой записи).

В последней части сообщения об ошибке сообщается:

```
The * operator expects to be given two Numbers
(Оператор * предполагает использование двух чисел)
```

Обратите особое внимание на слово «Numbers» («числа»). Pyret сообщает вам, какой тип информации работает в операции *. В программировании все значения организованы по типам (types) (например, число, строка, изображение). Эти типы, в свою очередь, применяются для описания типов входных данных и результатов (т. е. выходных данных), с которыми работает любая функция. Например, для операции умножения * ожидается предоставление двух чисел, в результате чего возвращается число. В последнем выражении, приведенном выше, мы попытались нарушить это правило, поэтому Pyret вывел сообщение об ошибке.

Фраза о «нарушении правила» звучит почти как юридическая формулировка, не так ли? Это действительно так, и термин «контракт» (contract) означает требуемые типы входных данных и предполагаемые типы выходных данных при использовании конкретной функции. Ниже приведено несколько примеров контрактов Pyret (записанных в той нотации, которую вы увидите в документации):

```
* :: (x1 :: Number, x2 :: Number) -> Number

circle :: (radius :: Number,
           mode :: String,
           color :: String) -> Image

rotate :: (degrees :: Number,
           img :: Image) -> Image

overlay :: (upper-img :: Image,
            lower-img :: Image) -> Image
```

> ***Выполните прямо сейчас***
>
> Внимательно рассмотрите примеры форм записи этих контрактов. Можете ли вы выделить различные части? Какую информацию они предоставляют вам?

Рассмотрим подробнее контракт `overlay`, чтобы убедиться в том, что вы понимаете, как правильно прочитать его. Этот контракт предоставляет несколько элементов информации:

- эта функция называется `overlay`;
- она принимает два элемента входных данных (части в круглых скобках) – оба имеют тип `Image` (изображение);
- первый элемент входных данных – изображение, которое будет расположено сверху;
- второй элемент входных данных – изображение, которое будет расположено снизу;
- вывод результата вызова этой функции (который следует за `->`) будет иметь тип `Image` (изображение).

В общем случае мы читаем два символа двоеточия (::) как «имеет тип», а составную стрелку (->) – как «возвращает».

Когда вы объединяете небольшие выражения в более сложные, типы результатов исходных выражений должны соответствовать типам, требуемым функцией, которая используется для их объединения. В приведенном выше ошибочном выражении с операцией умножения по контракту для * ожидаются два числа как входные данные, но мы передали изображение как второй элемент ввода. Поэтому при попытке выполнения этого выражения получили сообщение об ошибке.

Контракт также позволяет узнать, сколько элементов входных данных ожидает функция. Рассмотрим контракт для функции `circle`. Она ожидает три элемента входных данных: число (радиус), строку (стиль) и еще одну строку

(цвет). А если мы забыли строку стиля и передали только радиус и цвет, как показано ниже:

```
circle(100, "purple")
```

В этом случае ошибка не связана с типом входных данных, она относится к некорректному числу элементов переданных входных данных.

Упражнение 3.5

Выполните несколько выражений в Pyret, в которых используется некорректный тип некоторых входных данных для функции. В других выражениях передавайте неправильное количество входных данных в функцию.

Какой текст является общим для сообщений об ошибках, относящихся к некорректным типам данных? Какой текст является общим для сообщений об ошибках, относящихся к неправильному количеству входных данных?

Запишите этот общий текст, чтобы вы могли распознать аналогичные ошибки, если они возникнут во время программирования.

3.7.2. Ошибки формата и нотации

Мы только что наблюдали два различных типа ошибок, которые мы можем сделать при программировании: указание неверного типа входных данных и передача неправильного количества входных данных в функцию. Вероятно, вы также встречали еще один дополнительный тип ошибки – при некорректной расстановке знаков пунктуации при программировании. Для наглядности можно ввести в онлайновом редакторе следующие ошибочные примеры:

- `3+7`
- `circle(50 "solid" "red")`
- `circle(50, "solid, "red")`
- `circle(50, "solid," "red")`
- `circle 50, "solid," "red")`

> ***Выполните прямо сейчас***
>
> Убедитесь в том, что вы можете точно определить ошибку в каждом из приведенных выше примеров. Если необходимо, выполните их в Pyret.

Вам уже известны различные правила пунктуации при записи обычного текста. Для исходного кода также существуют свои правила пунктуации, и инструментальные средства программирования строго относятся к их соблюдению. В эссе произвольной формы вы можете пропустить запятую

без каких-либо проблем, но любая среда программирования не сможет выполнить выражения, если в них есть пунктуационные ошибки.

> ***Выполните прямо сейчас***
>
> Составьте список правил пунктуации для кода Pyret, которые, по вашему мнению, уже встречались вам.

Ниже приведена наша версия такого списка:

- арифметические операторы обязательно должны быть выделены пробелами с обеих сторон;
- для определения порядка выполнения операций требуются круглые скобки;
- при использовании функции пара круглых скобок окружает входные данные, элементы которых должны разделяться запятыми;
- если в начале строки используется двойная кавычка, то для завершения этой строки необходима еще одна двойная кавычка.

В программировании мы используем термин «синтаксис» (syntax) для обозначения правил записи корректных выражений (мы не сказали напрямую «правила пунктуации», потому что эти правила выходят за рамки того, что вы считаете пунктуацией, но это подходящий момент для ввода нового термина). Для начинающих ошибки в синтаксисе – вполне обычное дело. Со временем вы усвоите все эти правила. А сейчас не расстраивайтесь, если получаете сообщения о синтаксических ошибках от Pyret. Это одна из неотъемлемых частей процесса обучения.

3.7.3. Поиск других функций: документация

В настоящий момент у вас, возможно, возник вопрос: что еще можно сделать с изображениями? Ранее упоминалось масштабирование изображений. Какие еще фигуры можно создать? Существует ли где-то список всех возможных действий, которые можно выполнять с изображениями?

К каждому языку программирования прилагается документация (documentation), в которой вы можете найти разнообразные доступные операции и функции, а также варианты конфигурирования их параметров. Документация может показаться ошеломляюще огромной для начинающих программистов, потому что содержит множество подробностей, о необходимости использования которых вы даже не знали раньше. Рассмотрим, как может воспользоваться документацией начинающий программист.

Откройте раздел документации Pyret о работе с изображениями Pyret Image Documentation (https://www.pyret.org/docs/latest/image.html). Обратите особое внимание на боковую панель слева. В ее верхней части можно видеть список всех тем, описанных в этом разделе документации. Выполняйте прокрутку до тех пор, пока не увидите в боковой панели термин «rectangle»: его окружают имена других функций, которые можно использовать для создания

разнообразных фигур. Выполните прокрутку еще немного ниже и увидите список функций для объединения и обработки изображений.

Если щелкнуть левой кнопкой мыши по имени фигуры или функции, то в большой правой панели выводится подробная информация об использовании этой конкретной функции. В затененной панели вы увидите ее контракт, ниже – описание того, что делает эта функция, затем конкретный пример или даже два, демонстрирующий ввод входных данных для практического применения этой функции (с результатом ее выполнения). Можно скопировать любой из этих примеров и вставить его в интерактивный редактор Pyret, чтобы посмотреть, как он работает (например, можно изменять входные данные).

В данный момент вся необходимая вам документация находится в разделе о работе с изображениями. В дальнейшем мы будем более глубоко изучать Pyret и его документацию.