ПРИЛОЖЕНИЕ

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РЯЗАНСКИЙ ГОСУДАРСТВЕННЫЙ РАДИОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
ИМЕНИ. В.Ф. УТКИНА

Кафедра «Вычислительная и прикладная математика»

ОЦЕНОЧНЫЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ

«Функциональное программирование»

Направление подготовки

09.03.04 «Программная инженерия»

Направленность (профиль) подготовки

«Программная инженерия»

Уровень подготовки – бакалавриат

Квалификация выпускника – бакалавр

Форма обучения – очная

Рязань

1 ОБЩИЕ ПОЛОЖЕНИЯ

Оценочные материалы – это совокупность учебно-методических материалов и процедур, предназначенных для оценки качества освоения обучающимися данной дисциплины как части основной образовательной программы.

Цель – оценить соответствие знаний, умений и уровня приобретенных компетенций, обучающихся целям и требованиям основной образовательной программы в ходе проведения текущего контроля и промежуточной аттестации.

Основная задача – обеспечить оценку уровня сформированности общекультурных и профессиональных компетенций и индикаторов их достижения, приобретаемых обучающимся в соответствии с этими требованиями.

Контроль знаний обучающихся проводится в форме текущего контроля и промежуточной аттестации.

Текущий контроль успеваемости и промежуточная аттестацияя проводятся с целью определения степени усвоения учебного материала, своевременного выявления и устранения недостатков в подготовке обучающихся и принятия необходимых мер по совершенствованию методики преподавания учебной дисциплины, организации работы обучающихся в ходе учебных занятий и оказания им индивидуальной помощи.

К контролю текущей успеваемости относятся проверка знаний, умений и навыков обучающихся на практических занятиях и лабораторных работах, по результатам выполнения и защиты обучающимися индивидуальных заданий, по результатам выполнения контрольных работ и тестов, по результатам проверки качества конспектов лекций и иных материалов.

В качестве оценочных средств на протяжении семестра используется устные и письменные ответы студентов на индивидуальные вопросы, письменное тестирование по теоретическим разделам курса. Дополнительным средством оценки знаний и умений студентов является отчет о выполнении практических и лабораторных работ и их защита.

После пятого семестра обучающиеся сдают зачет. Форма проведения – устный ответ с письменным подкреплением по утвержденным билетам, сформулированным с учетом содержания дисциплины.

2 ОПИСАНИЕ ПОКАЗАТЕЛЕЙ И КРИТЕРИЕВ ОЦЕНИВАНИЯ КОМПЕТЕНЦИЙ

Сформированность каждой компетенции в рамках освоения данной дисциплины оценивается по трехуровневой шкале:

1) пороговый уровень является обязательным для всех обучающихся по завершении освоения дисциплины;

2) продвинутый уровень характеризуется превышением минимальных характеристик сформированности компетенций по завершении освоения дисциплины;

3) эталонный уровень характеризуется максимально возможной выраженностью компетенций и является важным качественным ориентиром для самосовершенствования.

Уровень освоения компетенций, формируемых дисциплиной

а) описание критериев и шкалы оценивания тестирования:

Шкала оценивания	**Критерий**
3 балла (эталонный уровень)	уровень усвоения материала, предусмотренного программой: процент верных ответов на тестовые вопросы от 85 до 100%
2 балла (продвинутый уровень)	уровень усвоения материала, предусмотренного программой: процент верных ответов на тестовые вопросы от 75 до 84%
1 балл (пороговый уровень)	уровень усвоения материала, предусмотренного программой: процент верных ответов на тестовые вопросы от 60 до 74%
0 баллов	уровень усвоения материала, предусмотренного программой: процент верных ответов на тестовые вопросы от 0 до 59%

б) описание критериев и шкалы оценивания теоретического вопроса:

Шкала оценивания	**Критерий**
3 балла (эталонный уровень)	выставляется студенту, который дал полный ответ на вопрос, показал глубокие систематизированные знания, смог привести при-

	меры, ответил на дополнительные вопросы преподавателя.
2 балла (продвинутый уровень)	выставляется студенту, который дал полный ответ на вопрос, но на некоторые дополнительные вопросы преподавателя ответил только с помощью наводящих вопросов.
1 балл (пороговый уровень)	выставляется студенту, который дал неполный ответ на вопрос в билете и смог ответить на дополнительные вопросы только с помощью преподавателя.
0 баллов	выставляется студенту, который не смог ответить на вопрос

в) описание критериев и шкалы оценивания практического задания:

Шкала оценивания	Критерий
3 балла (эталонный уровень)	Задание решено верно
2 балла (продвинутый уровень)	Задание решено верно, но имеются технические неточности в выполнении
1 балл (пороговый уровень)	Задание решено верно, с дополнительными наводящими вопросами преподавателя
0 баллов	Задание не решено

На зачет выносится два теоретических вопроса и одна задача.

Студент может набрать максимум 9 баллов.

Итоговый суммарный балл студента, полученный при прохождении промежуточной аттестации, переводится в традиционную форму по системе «зачтено», «не зачтено».

Оценка «зачтено» выставляется студенту, который набрал в сумме не менее 5 баллов. Обязательным условием является выполнение всех предусмотренных в течение семестра практических заданий и лабораторных работ.

Оценка «не зачтено» выставляется студенту, который набрал в сумме менее 5 баллов, либо имеет к моменту проведения промежуточной аттестации несданные практические, либо лабораторные работы.

Шкала оценивания	Критерий	
отлично (эталонный уровень)	8 – 9 баллов	Обязательным условием является выполнение всех предусмотренных в течение семестра практических заданий и лабораторных работ.
хорошо (продвинутый уровень)	6 – 7 баллов	
удовлетворительно (пороговый уровень)	4 – 5 баллов	
неудовлетворительно	0 – 3 баллов	Студент не выполнил всех предусмотренных в течение семестра текущих заданий

3 ПАСПОРТ ОЦЕНОЧНЫХ МАТЕРИАЛОВ ПО ДИСЦИПЛИНЕ

Контролируемые разделы (темы) дисциплины	Код контролируемой компетенции (или её части)	Наименование оценочного средства
Раздел 1. Введение в функциональное программирование		
Императивное и функциональное программирование	ПК-1.3, ПК-3.1, ПК-3.2	Зачет
История и перспективы развития функционального программирования	ПК-1.3, ПК-3.1, ПК-3.2	Зачет
Раздел 2. Теоретические основы функционального программирования		
Основы лямбда-исчисления	ПК-1.3, ПК-3.1, ПК-3.2	Зачет
Чистое лямбда-исчисление, как язык программирования	ПК-1.3, ПК-3.1, ПК-3.2	Зачет

Раздел 3. Основы программирования на функциональном языке Haskell		
Синтаксические конструкции и основные типы данных Haskell	ПК-1.3, ПК-3.1, ПК-3.2	Зачет
Техника построения рекурсивных программ в Haskell	ПК-1.3, ПК-3.1, ПК-3.2	Зачет
Модули	ПК-1.3, ПК-3.1, ПК-3.2	Зачет
Типы, определяемые пользователем	ПК-1.3, ПК-3.1, ПК-3.2	Зачет
Принципы организации ввода-вывода в языке Haskell	ПК-1.3, ПК-3.1, ПК-3.2	Зачет
Раздел 4. Средства функционального программирования в императивных языках		
Функциональное программирование на языке Python	ПК-1.3, ПК-3.1, ПК-3.2	Зачет
Средства функционального программирования в современных языках программирования	ПК-1.3, ПК-3.1, ПК-3.2	Зачет

4 ТИПОВЫЕ КОНТРОЛЬНЫЕ ЗАДАНИЯ ИЛИ ИНЫЕ МАТЕРИАЛЫ

4.1 Промежуточная аттестация (зачет, экзамен)

ПК-1: Способен разрабатывать требования, проектировать и выполнять программную реализацию программного обеспечения
ПК-1.3. Проектирует программное обеспечение и выполняет его программную реализацию

а) типовые тестовые вопросы закрытого типа

1. Стандартизированный чистый функциональный язык программирования общего назначения:
   Haskell
   Scala
   Ruby
2. Haskell является одним из самых распространённых языков программирования с поддержкой таких вычислений:
   дополнительных
   отложенных
   второстепенных
3. Что является основной управляющей структурой в функциональном языке?
   матрица
   таблица
   функция
4. В этом году была предложена первая версия языка, Haskell 0:
   1995
   1990
   1998
5. Одна из основных характеристик языка Haskell:
   всегда полное применение
   дополнительное применение
   частичное применение
6. Одна из основных характеристик языка Haskell:
   ленивые вычисления
   резкие вычисления
   мгновенные вычисления
7. Одна из основных характеристик языка Haskell:
   не сопоставление с образцом
   мгновенные вычисления
   сопоставление с образцом
8. За счет чего функциональные программы содержат меньше ошибок?

функциональные программы не содержат побочных эффектов
функциональные программы короче
на функциональных языках автоматически контролируются ошибки типа переполнения буфера
функциональные программ более просты и понятны для программиста

9. Как реализуются повторяющиеся действия в функциональных языках?
с помощью рекурсии
с помощью конструкций циклов
с помощью оператора перехода
с помощью функциональной абстракции
10. Какой принцип построения функциональных программ?
программа строится из набора функций, каждая из которых перерабатывает входные данные в выходные. Функции также могут рассматриваться как данные
программа строится из набора функций, каждая из которых перерабатывает входные данные в выходные. Существует четкое разделение между данными и функциями
программа строится из набора вызывающих друг друга подпрограмм (процедур и функций)
программа представляет собой одно большое арифметическое выражение
11. Какие основные способы борьбы со сложностью используются в функциональных программах?
функциональная абстракция и функциональная декомпозиция
наследование и полиморфизм
функциональная абстракция и мемоизация
функциональная декомпозиция и динамическое связывание
12. Какие языки программирования являются преимущественно функциональными?
C++
Java
C#
Haskell
F#
FORTH
Objective C
13. Какая алгоритмическая модель лежит в основе функционального программирования?
λ-исчисление
логика предикатов 1-го порядка
логика высших порядков
машина Тьюринга
14. Какая алгоритмическая модель лежит в основе императивного программирования?
λ-исчисление
логика предикатов 1-го порядка
логика высших порядков
машина Тьюринга
15. В чем отличия функционального программирования и императивного?
функциональное программирование оперирует функциями и их применением к данным, императивное – операторами и тем, как они изменяют состояние памяти
в фунциональном программировании каждая функция может оперировать только с той областью памяти, которая для нее выделена
в функциональном программировании происходит автоматический поиск решения задачи по ее декларативному описанию
все вышеперечисленное
16. Почему функциональное программирование сейчас представляет повышенный интерес для изучения?
это молодое направление программирования
многие программные проекты сейчас реализуются на функциональных языках
функциональный подход помогает решать такие проблемы, как распараллеливание вычислений
17. За счет чего функциональные программы обычно содержат меньше ошибок, чем императивные?
они короче, поэтому меньше шанс ошибиться

программисты на функциональных языках обычно умнее, поэтому делают меньше ошибок
функциональные программы проще отлаживать из-за их естественной модульности
функциональные программы не содержат побочных эффектов

18. Почему функциональные программы не содержат побочных эффектов?
    отсутствует понятие переменной и оператора присваивания
    функция может оперировать только над переменными, описанными внутри нее
    запрещено модифицировать внутренние переменные функции извне самой функции
    отсутствует понятие области видимости
19. Комментарии в Haskell обозначаются:
    /*текст*/
    {текст}
    {-текст-}
    //текст
20. Идентификатором переменной в Haskell является (укажите все правильные варианты):
    name
    naMe
    Name
    NaMe
21. В каком примере записан строковый литерал в Haskell?
    "текст"
    'текст'
    Текст
22. Решение какой задачи описывает программа:

```
length  ::[a] → Integer
length    [] = 0
length  (x:xs) = 1 + length xs
```

    сложение элемента к списку
    приписывает к списку 1
    подсчет количества элементов в списке
23. Какие из перечисленных идентификаторов являются зарезервированными (укажите все правильные варианты):
    case
    import
    default
    deriving
    infixl
24. Префиксная запись находится в выражении:
    x 'op' y
    x + y
    (-) x y
    x y (+)
25. Инфиксная запись находится в выражении (укажите все правильные варианты):
    (+) x y
    'op' x y
    x 'op' y
    x (+) y
    x y 'op'

б) типовые тестовые вопросы открытого типа

1. Сколько уровней приоритета имеет Haskell? (***Ответ***: от 0 до 9)
2. С какого ключевого слова начинается объявление модуля в Haskell? (***Ответ***: `module`)
3. Какое зарезервированое слово указывает на импортирование элемента в Haskell? (***Ответ***: `import`)
4. Какое ключевое слово используется для указания квалифицированного имени в Haskell? (***Ответ***: `qualified`)

5. Какой ответ даст при запросе `square 5` следующая функция:

```
square :: Integer → Integer
square x = x*x
```

(***Ответ***: 25)

6. Какой ответ выдаст Haskell на команду: `[1,2] ++ [3,4]`? (***Ответ***: [1,2,3,4])
7. Какой тип в Haskell означает целые типы фиксированной точности? (***Ответ***: Int)
8. Какой ответ выдаст Haskell на команду: `tail [1,2,3,4]`? (***Ответ***: [2, 3, 4])
9. Какой ответ выдаст Haskell на команду: `head (tail [1,2,3,4])`? (***Ответ***: 2)
10. Какой тип аргументов у функции `ord` в Haskell? (***Ответ***: Char)
11. Чему равно значение выражения, записанного в Haskell: `(*) 2 ((+) 1 4 ) ** 2`. (***Ответ***: 100)
12. Чему равно значение выражения, записанного в Haskell: `1 + 3 *+* 2 * 2`, где оператор *+* определен следующим образом:

```
infixl 6 *+*
a *+* b = a ^ 2 + b ^ 2
```

(***Ответ***: 32)

13. Чему равно значение выражения, записанного в Haskell: `89 |-| 200`, где оператор `|-|` определен следующим образом:

```
infixl 6 |-|
(|-|) a b = if a > b then a - b else b - a
```

(***Ответ***: 111)

14. Что определяет команда: `sum [n^2| n <- [1..100], (mod n 3) == 0 || (mod n 5) == 0]`? (***Ответ***: сумму квадратов чисел от 1 до 100, которые кратны 3 или 5)
15. Какой ответ выдаст Haskell на команду: `drop 3 [1, 3, 4, 5, 6, 7, 3]`? (***Ответ***: [5, 6, 7, 3])

ПК-3: Способен разрабатывать компоненты системных программных продуктов
ПК-3.1. Разрабатывает системные утилиты программного обеспечения
ПК-3.2. Создает компоненты инструментальных средств программирования

а) типовые тестовые вопросы закрытого типа

1. Во что заключается инфиксный оператор в Haskell?
   - кавычки
   - апострофы
   - ***обратные кавычки***
2. В Haskell оператор: infixr 5 ++ является:
   - левоассоциативными
   - ассоциативными
   - ***правоассоциативными***
3. Как будет интерпретироваться выражение `f x + g y` при разборе? Выберите правильный вариант:
   - `f x (+) g y`
   - ***`(f x) + (g y)`***
   - `f (x + g) y`
   - `f (x) + g (y)`
4. Каким значением определяются ошибки?
   - `error`
   - `warning`
   - `#116`
   - ***⊥***
5. В каком из примеров записан список в Haskell?
   - `(1,2,3..45)`
   - ***`['a','b','c','d']`***
   - `{1,2,'a','b','c'}`
   - `List ('a','b','c')`

6. В каком примере записан кортеж в Haskell?
   `[a,b,c]`
   `(a,b,c)`
   `{a,b,c}`
   `cor (a,b,c)`
7. В каком примере правильно объявлен новый тип данных:
   `String Color = Red|Green|Yellow`
   `Char Color = Red|Green|Yellow`
   `data Color = Red|Green|Yellow`
8. Какие опреации Haskell определяет класс Eq:
   `== и /=`
   `=> и />`
   `=< и =`
   `>,< и =`
9. `infixl` задает:
   правило вычисления целых чисел
   преоритет операторов
   условие завершения цикла
10. Модули в Haskell используются для:
    управления пространтсвами имен и для создания абстратных типов данных
    описания отдельных функции работ и в других программах
    создание новых компонентов
11. Что представляет собой запись A.x в Haskell:
    квалифицированное имя
    использование инфиксной функции
    неквалифицированное имя
12. Посредством какой формы можно исключить сущности в Haskell
    `module modid()`
    `module Class (modid())`
    `hiding (import1,...,importn)`
13. Задано объявление импорта `import qualified A as B`. Какое имя будет в области видимости?
    `A.x`
    `A.x, B.x`
    `B.x, B.y`
    `.x, B.y`
14. Запись вида (Int, Bool, Int) в Haskell означает:
    кортеж
    список
    функцию
15. Экземпляром класса `Functor` в Haskell является тип:
    Ordering
    Overload
    maybe
16. Выберите те числовые типы, которые используются в языке Haskell:
    Real
    Int
    Int64
    Integer
    Currency
    Extended
17. Каким типом представлено исключение в монаде ввода-вывода:
    Warning
    Error
    IOWarning
    IOException
    IOError

18. Какие из ниже перечисленных функций относятся к функциям над списками:
    head
    repeat
    until
    zipcode
    zipwidth
    unzip
19. Какая из этих функций может возвращать бесконечный список?
    iterate
    cycle
    repeat
    replicate
20. Не используя GHCi, выберите выражения, проходящие проверку типов (выберите все подходящие ответы из списка):
    `[1,2] : 3 ++ [4,5,6]`
    `[1,2] ++ [3,4,5] : 6`
    `1 : [2,3] ++ [4,5,6]`
    `(++) [1,2] 3 : [4,5,6]`
    `(:) 1 ((++) [2,3] [4,5,6])`
    `[1,2] ++ 3 : [4,5,6]`
    `(:) 1 (++) [2,3] [4,5,6]`
    `[1,2] ++ (:) 3 [4,5,6]`
21. Аналог какой функции по работе со списками (Haskell) приведен в коде?

```
func :: [a] -> a
func [] = error "Error"
func (x:_) = x
```

    head
    tail
    last
    first
22. Аналог какой функции по работе со списками (Haskell) приведен в коде?

```
func :: (a -> b) -> [a] -> [b]
func _ [] = []
func f (x:xs) = f x : func f xs
```

    tail
    map
    filter
    zip
23. Аналог какой функции по работе со списками (Haskell) приведен в коде?

```
func :: (a -> Bool) -> [a] -> [a]
func _ [] = []
func p (x:xs)
    | p x = x : func p xs
    | otherwise = []
```

    dropWhile
    map
    takeWhile
    filter
24. Аналог какой функции по работе со списками (Haskell) приведен в коде?

```
func :: Num a => [a] -> Int
func []     = 0
func (x:xs) = 1 + func xs
```

    len

sum
tail
head

25. Что делает декоратор (Python):

```
def decor(func):
    def wrapped(*args, **kwargs):
        return "<i>" + func(*args, **kwargs) + "</i>"
    return wrapped

@decor
def text(s):
    return s

st = input()
print(text(st))
```

отображает текст курсивом
добавляет перед строкой st тег «», а после строки st тег «»
добавляет перед и после строки «»

26. Пусть в программе на Python имеется некоторый генератор `gen`, который возвращает числа. Отметьте команды, которые можно выполнять с этим генератором (Выберите все подходящие ответы из списка).
`sum(gen)`
`gen[0]`
`min(gen)`
`list(gen)`
`set(gen)`
`tuple(gen)`
`max(gen)`
`len(gen)`

27. Как называется преобразование функции от многих аргументов в набор функций, каждая из которых является функцией от одного аргумента?
каррирование
упрощение
кадрирование
квантование
расщепление

28. Что такое функция-декоратор?
Функция, которая оборачивает другую функцию для расширения ее функционально-сти
Функция, которая служит для улучшения читаемости кода, но не влияет на его функционал явно
Функция, которая задает внешний вид программы/сайта
Функция из модуля decorator, которая нужна для подсветки синтаксиса?

29. Сколько строк кода можно вписать в *lambda*-функцию
Бесконечно много
Одну
Две
Лямбда-функция принимает только импортированные функции?

30. Для чего служит функция `reduce`?
Для уменьшения значения всех элементов коллекции на переданное значение
Для применения функции ко всем элементам последовательности
Для комбинирования всех значений в наборе определенной функцией
Для слияния всех значений в наборе в одно целое?

б) типовые тестовые вопросы открытого типа

1. Какую задачу решает функция (Haskell):

```
func :: Integer -> Integer
func 0 = 0
func n = mod n 10 + func (div n 10)
```

(***Ответ***: определяет сумму цифр числа)

2. Какую задачу решает функция (Haskell):

```
func :: Ord a => [a] -> a -> [a]
func p k = filter (\x -> x <= k) p
```

(***Ответ***: удаляет из списка элементы, большие заданного числа *k*)

3. Какую задачу решает функция (Haskell):

```
func :: [Double] -> Double
func p = sum (take 10 p)
```

(***Ответ***: возвращает сумму первых 10 элементов списка)

4. Какую задачу решает функция (Haskell):

```
func :: Num a => [a] -> a
func [] = 0
func p = product $ map (\x -> x*2 + 3) p
```

(***Ответ***: каждый элемент списка умножает на 2, потом прибавляет 3 и возвращает произведение всех элементов)

5. Какую задачу решает функция (Haskell):

```
func :: [Double] -> Double
func p = sum p / fromIntegral (length p)
```

(***Ответ***: находи среднее арифметическое элементов списка вещественных чисел)

6. Какую задачу решает функция (Haskell):

```
func :: [Integer] -> [Integer]
func [] = []
func (x:xs) = (x+1) : func xs
```

(***Ответ***: увеличивает каждый элемент списка из целых чисел на единицу)

7. Какую задачу решает функция (Haskell):

```
func :: (a -> Bool) -> [a] -> [a]
func _ [] = []
func p (x:xs)
    | p x = x : func p xs
    | otherwise = []
```

(***Ответ***: возвращает элементы списка, которые удовлетворяют предикату)

8. Какую задачу решает функция (Haskell):

```
func :: a -> a -> [a] -> [a]
func x y z = x : y : z
```

(***Ответ***: добавляет два переданных ей значения в голову переданного списка)

9. Какую задачу решает функция (Haskell):

```
func :: Num a => [a] -> Int
func []     = 0
func (x:xs) = 1 + func xs
```

(***Ответ***: определяет количество элементов в переданном в функцию списке)

10. Какую задачу решает функция (Haskell):

```
func :: Ord a => [a] -> [a]
func [] = []
func (x:xs) = func [y | y <- xs, y <= x] ++ [x] ++ func [y | y <- xs, y > x]
```

(***Ответ***: сортирует список по возрастанию)

11. Напишите результат работы скрипта (Python):

```
def even_range(start, end):
    curr = start
    while curr < end:
        yield curr
        curr += 2

for num in even_range(-5, 6):
    print(num, end=' ')
```

(***Ответ***: `-5 -3 -1 1 3 5`)

12. Напишите результат работы скрипта (Python):

```
def squarify(n):
    return n ** 2

ls = list(range(5))
print(*map(squarify, ls))
```

(***Ответ***: `0 1 4 9 16`)

13. Напишите результат работы скрипта (Python):

```
def only_pos(x):
    return x ** 0.5 == int(x ** 0.5)

result = filter(only_pos, [28, 25, 66, 64, 225])
print(", ".join(str(x) for x in result))
```

(***Ответ***: `25, 64, 225`)

14. Напишите результат работы скрипта (Python):

```
animals = ['лев', 'кролик', 'крыса', 'свинья']

ls_len = list(map(len, animals))
print(*ls_len)
```

(***Ответ***: `3 6 5 6`)

15. Напишите результат работы скрипта (Python):

```
from functools import reduce

print(reduce(lambda a, x: a + x, [0, 1, 2, 3, 4]))
```

(***Ответ***: 10)

16. Как называется прием, когда в теле функции присутствуют ссылки на переменные, объявленные вне тела этой функции в окружающем коде и не являющиеся её аргументами? (***Ответ***: замыкание)
17. Как в Python называются функции-обёртки, которые дают возможность изменить поведение функции, не изменяя её код? (***Ответ***: декораторы)
18. Напишите результат работы скрипта (Python):

```
ls = [31, 23, 32, 332, 33, 51, 45, 103, 36 ]
ans = filter(lambda x: x % 2 != 0, filter(lambda n: '3' in str(n), ls))
print(*list(ans))
```

(***Ответ***: `31 23 33 103`)

4.2 Типовые вопросы к зачету по дисциплине (5-й семестр)

1. Энергичные и ленивые вычисления.
2. Императивное и функциональное программирование: основные идеи, особенности. Преимущества и недостатки функционального программирования.
3. Функциональные эквиваленты императивных программ.
4. Строго функциональный язык.
5. История возникновения и развития функционального программирования.
6. Теоретическая база функционального программирования: рекурсивные функции, *Лямбда*-исчисление *А.Чёрча*.
7. Основные принципы работы в Haskell Platform. Команды интерпретатора для работы с файлами программ. Пример создания первой программы.
8. Модель вычислений. Применение функций и скобки. Частичное применений функций.
9. Понятие функции. Определение и вызов функции. Чистота функций. Тип определяемой функции. Префиксная постфиксная нотация.
10. Определение функции через частичное применение. Пример.
11. Условное выражение в языке *Haskell*. Пример.
12. *Haskell*. Операторы и функции. Приоритеты и ассоциативность операторов. Реализация собственных операторов.
13. *Haskell*. Сечение операторов. Оператор применения с низким приоритетом.
14. *Haskell*. Функции высших порядков. Каррирование функции. Частичное применение. Пример программы.
15. Вывод типа выражения. Числовые типы и числовые литералы. Основные типы языка *Haskell*.
16. *Haskell*. Тип функции. Допустимые имена переменных и функций.
17. *Haskell*.Классы типов. Полиморфные типы. Синонимы типов.
18. *Haskell*. Тип кортежа. Функции для работы с кортежами.
19. *Haskell*. Тип списка. Функции для работы со списками.
20. *Haskell*. Генераторы списков. Примеры использования.
21. *Haskell*. Функции высших порядков для работы со списками.
22. *Haskell*. `Case`-выражения. Синтаксис и примеры использования.
23. *Haskell*. Рекурсия. Пример реализации.
24. *Haskell*. Сопоставление с образцом. Понятие клоза, неопровержимого образца. Пример программы.
25. *Haskell*. Образец вида (`x:xs`). Пример использования.
26. *Haskell*. Именованные образцы. Пример использования.
27. *Haskell*. Сигнализация об ошибках. Не завершающиеся программы, использование `error` и `undefined`.
28. *Haskell*. Охранные выражения. Пример использования.
29. *Haskell*. Собственная реализация функций для работы со списками `zip`, `zipWith`, `map`, `filter`, `takeWhile` и др.
30. *Haskell*. Декларативный стиль определения функции. Пример программы.
31. *Haskell*. Композиционный стиль определения функции. Пример программы.
32. *Haskell*. Рекурсия с явным аккумулятором. Пример программы.
33. *Haskell*. Синтаксис записи лямбда-выражений в *Haskell* и их использование.
34. *Haskell*. Использование `let`-выражений в локальной области видимости. Правила выравнивания. Пример программы.
35. *Haskell*. Использование `where`-выражений в локальной области видимости. Правила выравнивания. Пример программы
36. *Haskell*. Использование `let`-выражений в локальной области видимости. Пример программы.
37. *Haskell*. Правая свертка. Ее реализация и пример использования.

38. *Haskell*. Левая свертка. Ее реализация и пример использования.
39. *Haskell*. Создание собственных модулей. Пример программы.
40. *Haskell*. Экспорт модулей.
41. *Haskell*. Использование квалифицированных имен.
42. *Haskell*. Типы перечислений. Реализация производных представлений. Пример создания пользовательских типов.
43. *Haskell*. Типы сумм произведений. Пример описания типа и его использование.
44. *Haskell*. Синтаксис записей. Пример описания и использования записей.
45. *Haskell*. Модификация записей. Пример программы.
46. *Haskell*. Синтаксис записей в сопоставлении с образцом. Пример программы.
47. *Haskell*. Рекурсивные типы. Списки как рекурсивные типы.
48. *Haskell*. Абстрактные типы данных. Класс типов `Functor`. Представитель класса типов `Functor` для бинарного дерева.
49. *Haskell*. Класс типов `Monad`. Оператор монадического связывания.
50. *Haskell*. Монада `Identity`.
51. *Haskell*. Монада `Maybe`. Пример использования монады `Maybe`.
52. *Haskell*. Операции ввода-вывода. Тип `IO`. Монада `IO`.
53. *Haskell*. Простейший `Reader`. Универсальный `Reader`.
54. *Haskell*. Монада `Writer`. Интерфейс для монады `Writer`.
55. *Haskell*. Создание исполняемых программ.
56. *Python*. Замыкание функций. Декораторы. Каррирование в Python. Пример реализации.
57. *Python*. Итераторы и генераторы, оператор `yield`. Пример программы.
58. *Python*. Итератор и итерируемые объекты. Функции `iter`() и `next`(). Пример программы.
59. *Python*.Выражения-генераторы и функции-генераторы. Пример программы.
60. *Python*. Анонимные функции в Python. Пример программы.
61. *Python*. Функции, позволяющие работать в функциональном стиле. Функция `map`().
62. *Python*. Функции, позволяющие работать в функциональном стиле. Функция `filter`()
63. *Python*. Функции `zip`(), `all`() и `any`().
64. *Python*. Модуль `itertools`.
65. *Python*. Модуль `functools`.

4.3 Типовые задачи на зачет по дисциплине (5-й семестр)

1. *Haskell*. Написать собственную реализацию функции `last` – определения последнего элемента списка.
2. *Haskell*. Написать собственную реализацию функции `init`, которая возвращает все элементы списка, кроме последнего.
3. *Haskell*. Написать функцию добавления элемента в конец списка.
4. *Haskell*. Написать функцию, осуществляющую реверс списка: первый элемент ставим последним, второй – предпоследним, и так далее.
5. *Haskell*. Реализуйте функцию, находящую значение определённого интеграла от заданной функции на заданном интервале [*a*; *b*] методом трапеций. (Используйте деление отрезка интегрирования на *n* элементарных отрезков.)
6. *Haskell*. Реализуйте функцию, находящую значение определённого интеграла от заданной функции на заданном интервале [*a*; *b*] методом прямоугольников. (Используйте деление отрезка интегрирования на *n* элементарных отрезков.)
7. *Haskell*. Используя метод накапливающего параметра реализуйте функцию `seqA`, находящую элементы рекуррентной последовательности: $a_0 = 1, \quad a_1 = -3, \quad a_2 = 2, \quad a_{k+3} = 2a_k + a_{k+1} - a_{k+2}$. Решение должно быть эффективным, а вспомогательная функция должно быть доступна только в функции `seqA`.

8. *Haskell*. Используя метод накапливающего параметра реализуйте функцию `seqA`, находящую элементы последовательности: $a_0 = -3, \quad a_1 = -2, \quad a_2 = 1, \quad a_{k+3} = a_k - 2a_{k+1} + a_{k+2}$ (рекуррентной). Решение должно быть эффективным, а вспомогательная функция должно быть доступна только в функции `seqA`.
9. *Haskell*. Реализуйте функцию, возвращающую элемент списка по его номеру.
10. *Haskell*. Функция `takeWhile` принимает предикат и список и возвращает те элементы списка, которые удовлетворяют предикату. Как только найдется элемент, не удовлетворяющий предикату, обход останавливается. Реализуйте функцию `takeWhile`.
11. *Haskell*. Написать функцию, которая из списка удаляет элементы, не являющиеся простыми числами.
12. *Haskell*. Реализовать поиск нуля функции одной переменной методом половинного деления на отрезке [*a*; *b*] (где функция принимает разные знаки).
13. *Python*. Вводится строка целых чисел через пробел. Напишите функцию, которая преобразовывает эту строку в список чисел и возвращает их сумму. Определите декоратор для этой функции, который имеет один параметр `start` –начальное значение суммы. Примените декоратор со значением `start=5` к функции и вызовите декорированную функцию. Результат отобразите на экране.
14. *Python*. Задайте функцию-генератор, которая на основе символов строки `chars`:

```
from string import ascii_lowercase
from string import ascii_uppercase
chars = ascii_lowercase + ascii_uppercase + "0123456789!?@#$*"
```

формирует и выдает случайно сгенерированные пароли длиной 12 символов. Количество выдаваемых паролей функцией должно быть неограниченным. Случайный выбор символа из последовательности `chars` можно реали-зовать с помощью функции `choice` модуля `random`. Выведите первые пять сгенерированных паролей на экран.
15. *Python*. Определите функцию-генератор, которая бы перебирала все комбинации из трех целых положительных чисел, каждое в диапазоне [2; 20]. Для каждой тройки чисел она должна определять, могут ли они образовывать длины сторон треугольника. (Критерий: длина любой стороны треугольника должна быть меньше суммы двух других). Функция должна возвращать текущие длины сторон и булево значение `True` – если числа образуют треугольник и `False` – в противном случае. Выведите с помощью этой функции-генератора первые 20 значений. Какие-либо коллекции в программе не использовать.