

Документ подписан простой электронной подписью
Информация о владельце:
ФИО: Гаскаев Сергей Валерьевич
Должность: Ректор
Дата подписания: 05.09.2025 11:15:02
Уникальный программный ключ:
04c19ed8bf98f3b6cb77a486b9a8788842574



МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Челябинский государственный университет» (ФГБОУ ВО «ЧелГУ»)

Фонд оценочных средств для промежуточной аттестации по дисциплине (модулю) «Современные языки программирования» по направлению подготовки 09.04.03 «Прикладная информатика» направленности «Управление решениями на основе данных» ФГБОУ ВО «ЧелГУ»

стр. 1

Фонд оценочных средств для промежуточной аттестации по дисциплине (модулю)
«Современные языки программирования»

Направление подготовки (специальность)
09.04.03 «Прикладная информатика»

Направленность (профиль)
«Управление решениями на основе данных»

Присваиваемая квалификация
Магистр

Форма обучения
Очная

Год набора
2025

Челябинск, 2025 г.

09.04.03 Прикладная информатика, Управление решениями на основе данных, магистр, *Современные языки программирования*, 2025, очная

Фонд оценочных средств дисциплины (модуля) одобрен и рекомендован

Проректор по учебной работе утверждено 24.02.2025 А.А. Саламатов

Ученым советом института информационных технологий

Протокол заседания № 6 от 20.02.2025

Председатель Ученого совета
института информационных
технологий

согласовано

Ю. В. Петриченко

Заседанием кафедры информационных технологий и экономической информатики

Протокол заседания № 6 от 20.02.2025

И. о. заведующего кафедрой

согласовано

С.А. Скрипов

Автор (составитель)

В.А. Мельников

Структура рабочей программы соответствует приказу ректора ФГБОУ ВО «ЧелГУ» от «13» апреля 2021 г. № 247-1



Содержание

1. Паспорт фонда оценочных средств	3
2. Перечень формируемых компетенций	4
3. Содержание оценочных средств по дисциплине	6
3.1. Виды оценочных средств	6
3.2. Содержание оценочных средств	8
4. Порядок проведения и критерии оценивания промежуточной аттестации	58
4.1. Порядок проведения промежуточной аттестации	58
4.2. Критерии оценивания промежуточной аттестации по видам оценочных средств	58
4.3. Результаты промежуточной аттестации и уровни сформированности компетенций.....	58



МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Челябинский государственный университет» (ФГБОУ ВО «ЧелГУ»)

Фонд оценочных средств для промежуточной аттестации по дисциплине (модулю) «Современные языки программирования» по направлению подготовки 09.04.03 «Прикладная информатика» направленности «Управление решениями на основе данных» ФГБОУ ВО «ЧелГУ»

стр. 3

1. Паспорт фонда оценочных средств

Направление подготовки: 09.04.03 Прикладная информатика

Направленность: Управление решениями на основе данных

Дисциплина: Современные языки программирования

Семестры: 1

Форма промежуточной аттестации: экзамен

Для оценивания результатов обучения используется балльно-рейтинговая система.



2. Перечень формируемых компетенций

Изучение дисциплины «Современные языки программирования» направлено на формирование компетенций, приведённых в 1.

Таблица 1. Результаты обучения по дисциплине.

Коды компетенции и согласно ФГОС (ОПОП ВО)	Содержание компетенций согласно ФГОС (ОПОП ВО)	Индикаторы достижения компетенции согласно ОПОП	Перечень планируемых результатов обучения по дисциплине
1	2	3	4
ОПК-2	Способен разрабатывать оригинальные алгоритмы и программные средства, в том числе с использованием современных интеллектуальных технологий, для решения профессиональных задач;	ОПК-2.1. Демонстрирует знание современных интеллектуальные технологии и технологий программирования, инструментальных средств, программно-технических платформ для решения профессиональных задач ОПК-2.2. Демонстрирует умения разрабатывать алгоритмические и программные решения с использованием интеллектуальных технологий и технологий программирования ОПК-2.3. Имеет практический опыт разработки программного обеспечения для решения профессиональных задач	Знать: Основы языка программирования Python Уметь: Разрабатывать программное обеспечение на языке Python Владеть: Навыками разработки программного обеспечения на языке Python
ОПК-5	Способен разрабатывать и модернизировать программное и аппаратное обеспечение информационных и автоматизированных систем;	ОПК-5.1. Знает виды, архитектуру современного программного и аппаратного обеспечение информационных и автоматизированных систем ОПК-5.2. Демонстрирует умения разрабатывать и модернизировать программное и аппаратное обеспечение исходя из заданных требований к информационным и автоматизированным системам ОПК-5.3. Имеет практический опыт разработки программного и аппаратного обеспечения	Знать: Основные правила программирования на языке Python Уметь: Разрабатывать программный код под поставленную задачу используя Python Владеть: Навыками разработки ПО используя Python



		информационных и автоматизированных систем для решения профессиональных задач	
ПК-4	Способен выполнять проектирование цифровых продуктов, информационных систем и сервисов с выбором и обоснованием вариантов реализации	ПК-4.1. Демонстрирует знание архитектуры и принципов проектирования баз данных, сервисно-ориентированный подход к проектированию компонентов и интерфейсов цифровых продуктов ПК-4.2. Демонстрирует умение выполнять проектирование баз данных и сервисов ПК-4.3. Имеет практический опыт разработки и экспертной оценки технических спецификаций на компоненты баз данных и протоколы взаимодействия между сервисами	Знать: Основные правила и методы проектирования информационных систем Уметь: Осуществлять проектирование и разработку архитектуры программной системы Владеть: Навыками разработки спецификаций компонентов систем и интерфейсов взаимодействия между ними



3. Содержание оценочных средств по дисциплине

3.1. Виды оценочных средств

Таблица 2. Виды оценочных средств.

№ п/п	Код компетенции/ планируемые результаты обучения	Контролируемые темы/ разделы	Наименование оценочного средства для текущего контроля	Наименование оценочного средства на промежуточной аттестации/№ задания
1	ОПК-2.1. Демонстрирует знание современных интеллектуальные технологии и технологий программирования, инструментальных средств, программно-технических платформ для решения профессиональных задач Знать: Основы языка программирования Python	Введение в синтаксис языка программирования Python Тестирование и отладка ПО Проектирование приложений на python Веб фреймворки на языке Python	Тестирование, проверка практического задания	Ответ на устный вопрос
2	ОПК-2.2. Демонстрирует умения разрабатывать алгоритмические и программные решения с использованием интеллектуальных технологий и технологий программирования Уметь: Разрабатывать программное обеспечение на языке Python	Введение в синтаксис языка программирования Python Тестирование и отладка ПО Проектирование приложений на python Веб фреймворки на языке Python	Тестирование, проверка практического задания	Ответ на устный вопрос
3	ОПК-2.3. Имеет практический опыт разработки программного обеспечения для решения профессиональных задач Владеть: Навыками разработки программного обеспечения на языке Python	Введение в синтаксис языка программирования Python Тестирование и отладка ПО Проектирование приложений на python Веб фреймворки на языке Python	Тестирование, проверка практического задания	Ответ на устный вопрос
4	ОПК-5.1. Знает виды, архитектуру современного программного и аппаратного обеспечение информационных и автоматизированных систем Знать: Основные правила программирования на языке Python	Введение в синтаксис языка программирования Python Тестирование и отладка ПО Проектирование приложений на python Веб фреймворки на языке Python	Тестирование, проверка практического задания	Ответ на устный вопрос
5	ОПК-5.2. Демонстрирует умения разрабатывать и модернизировать программное и аппаратное обеспечение исходя из заданных требований к	Введение в синтаксис языка программирования Python Тестирование и отладка ПО Проектирование приложений на python Веб фреймворки на языке Python	Тестирование, проверка практического задания	Ответ на устный вопрос



	информационным и автоматизированным системам Уметь: Разрабатывать программный код под поставленную задачу используя Python	Python		
6	ОПК-5.3. Имеет практический опыт разработки программного и аппаратного обеспечения информационных и автоматизированных систем для решения профессиональных задач Владеть: Навыками разработки ПО используя Python	Введение в синтаксис языка программирования Python Тестирование и отладка ПО Проектирование приложений на python Веб фреймворки на языке Python	Тестирование, проверка практического задания	Ответ на устный вопрос
7	ПК-4.1. Демонстрирует знание архитектуры и принципов проектирования баз данных, сервисно-ориентированный подход к проектированию компонентов и интерфейсов цифровых продуктов Знать: Основные правила и методы проектирования информационных систем	Введение в синтаксис языка программирования Python Тестирование и отладка ПО Проектирование приложений на python Веб фреймворки на языке Python	Тестирование, проверка практического задания	Ответ на устный вопрос
8	ПК-4.2. Демонстрирует умение выполнять проектирование баз данных и сервисов Уметь: Осуществлять проектирование и разработку архитектуры программной системы	Введение в синтаксис языка программирования Python Тестирование и отладка ПО Проектирование приложений на python Веб фреймворки на языке Python	Тестирование, проверка практического задания	Ответ на устный вопрос
9	ПК-4.3. Имеет практический опыт разработки и экспертной оценки технических спецификаций на компоненты баз данных и протоколы взаимодействия между сервисами Владеть: Навыками разработки спецификаций компонентов систем и интерфейсов взаимодействия между ними	Введение в синтаксис языка программирования Python Тестирование и отладка ПО Проектирование приложений на python Веб фреймворки на языке Python	Тестирование, проверка практического задания	Ответ на устный вопрос



Типовые задания, критерии и показатели оценивания в рамках текущего контроля представлены в рабочей программе дисциплины (модуля). Полные комплекты оценочных средств и контрольно-измерительных материалов хранятся на кафедре.

3.2. Содержание оценочных средств

База тестовых вопросов

№ п/п	Формулировка вопроса	Варианты ответов
1.	Конструктор класса задается методом с именем:	(1) <code>__new__</code> (2) <code>__init__</code> (3) <code>__construct__</code> (4) <code>new</code> (5) <code>init</code> (6) имя конструктора совпадает с именем класса
2.	Если в классе определен деструктор с двумя и более параметрами, то:	(1) будет сгенерирована ошибка, т.к. деструктор не может иметь более одного параметра (2) будет сгенерировано предупреждение, и такой деструктор должен вызываться только явно (3) не будет сгенерировано ни предупреждения, ни ошибки; при неявном вызове деструктора значение параметра будет равно <code>None</code> (4) предупреждения не будет сгенерировано, но такой деструктор должен вызываться только явно
3.	Если в классе определены два метода с одинаковыми именами и разными списками параметров, то:	(1) при выполнении скрипта будет сгенерирована ошибка (2) будет сгенерировано предупреждение, второе определение заменит первое (3) не будет сгенерировано ни предупреждения, ни ошибки; второе определение заменит первое (4) не будет сгенерировано ни предупреждения, ни ошибки; вызов того или иного метода будет зависеть от типа и количества указанных при вызове параметров (5) будет сгенерировано предупреждение; вызов того или иного метода будет зависеть от типа и количества указанных при вызове параметров
4.	Значением поля класса по умолчанию может являться	(1) значение переменной (2) константа (3) результат вызова функции (4) возможность указания значений полей по умолчанию в Python не предусмотрена
5.	В языке Python объектами являются:	(1) экземпляры классов и переменные (2) экземпляры классов, переменные и функции (3) экземпляры классов, классы и переменные (4) все типы данных
6.	Деструктор класса задается методом с именем:	(1) <code>__del__</code> (2) <code>__delete__</code> (3) <code>__destr__</code> (4) <code>__destruct__</code>
7.	Укажите результат выполнения скрипта: <pre>class Foo: def __init__(self): print('constructor', end=' ') self.__del__(self)</pre>	(1) <code>constructor</code> (2) <code>destructor</code> (3) <code>constructor destructor</code> (4) <code>destructor constructor</code> (5) скрипт не будет выполнен, т.к. код содержит



	<pre>def __del__(self): print ('destructor', end=' ') obj = Foo()</pre>	ошибки
8.	<p>Укажите результат выполнения скрипта:</p> <pre>x=0 class Foo: count=x def __init__(self): self.count+=1 def __del__(self): self.count+=1 obj = Foo() print (obj.count)</pre>	(1) 0 (2) 1 (3) 2 (4) пустая строка (5) скрипт не будет выполнен, т.к. код содержит ошибки
9.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo: count=0 def __init__(self): self.count+=1 obj = Foo() print (obj.count)</pre>	(1) 0 (2) 1 (3) пустая строка (4) скрипт не будет выполнен, т.к. код содержит ошибки
10.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo: def foo(self): print ('1') def __init__(self): print ('2') x = Foo()</pre>	(1) 1 (2) 2 (3) 2;1 (4) пустая строка (5) скрипт не будет выполнен, т.к. код содержит ошибки
11.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo: def foo(self): print ('foo') del self def __del__(self): print ('del') obj = Foo() obj.foo()</pre>	(1) foo (2) del (3) del foo (4) foo del (5) скрипт не будет выполнен, т.к. код содержит ошибки
12.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo: def __init__(self): print ('construct') del self obj = Foo() if obj: print ('exist')</pre>	(1) construct (2) exist (3) construct exist (4) скрипт не будет выполнен, т.к. код содержит ошибки
13.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo (object): obj=0 def __new__(cls,*dt,**mp): return object.__new__(cls,*dt,**mp).obj def __add__(self, x): return self.obj+2*x</pre>	(1) 0 (2) 1 (3) 2 (4) скрипт не будет выполнен, т.к. код содержит ошибки



	<pre>o = Foo() o+=1 print (o)</pre>	
14.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo (object): def __new__(cls,*dt,**mp): print ('new', end=' ') def __init__(self): print ('init', end=' ') o = Foo()</pre>	<p>(1) new (2) init (3) new init (4) init new (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>
15.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo (object): obj=0 def __new__(cls,*dt,**mp): print (cls, end=' ') return object.__new__(cls,*dt,**mp).obj def __init__(self): self.obj+=2 print (self, end=' ') def __str__(self, x): return obj o = Foo() print (o, end=' ')</pre>	<p>(1) 0 2 2 (2) 2 2 (3) <class '__main__.Foo'> 0 (4) <class '__main__.Foo'> 2 2 (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>
16.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo (object): obj=0 def __new__(cls,*dt,**mp): print ('1') return object.__new__(cls,*dt,**mp).obj def __init__(self): print ('2') o = Foo() print (type (o))</pre>	<p>(1) 1 2 <class '__main__.Foo'> (2) 2 <class '__main__.Foo'> (3) 1 <class 'int'> (4)) 2 1 <class 'int'> (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>
17.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo (object): obj=0 def __new__(cls,*dt,**mp): obj=1 def __init__(self): self.obj=2 o = Foo() print (o.obj)</pre>	<p>(1) 0 (2) 1 (3) 2 (4) скрипт не будет выполнен, т.к. код содержит ошибки</p>
18.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo (object): obj=0 def __new__(cls,*dt,**mp): cls.obj+=1 return object.__new__(cls,*dt,**mp) def __init__(self): self.obj=self+2 def __add__(self, x): return self.obj+2*x</pre>	<p>(1) 3 (2) 4 (3) 5 (4) 6 (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>



	<pre>o = Foo() print (o.obj)</pre>	
19.	<p>Укажите результат выполнения скрипта:</p> <pre>lst=[1] class Foo (object): lst.append(2) print (lst) ob1 = Foo() ob2 = Foo()</pre>	<p>(1) [1, 2] (2) [1, 2] [1, 2, 2] (3) [1, 2] [1, 2, 2] [1, 2, 2, 2] (4) скрипт не будет выполнен, т.к. код содержит ошибки</p>
20.	<p>Укажите результат выполнения скрипта:</p> <pre>lst=[] class Foo (object): obj=0 lst.append(0) ob1 = Foo() ob2 = Foo() print (lst)</pre>	<p>(1) [] (2) [0] (3) [0, 0] (4) [0, 0, 0] (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>
21.	<p>Укажите результат выполнения скрипта:</p> <pre>lst=[0] class Foo (object): lst+=[1] ob = Foo() print (lst, ob.lst)</pre>	<p>(1) [0] [0] (2) [0] [0, 1] (3) [0, 1] [0, 1] (4) скрипт не будет выполнен, т.к. код содержит ошибки</p>
22.	<p>Укажите результат выполнения скрипта:</p> <pre>foo = [1] class Foo (object): bar = foo bar += [1] ob = Foo() print (foo)</pre>	<p>(1) [1, 1] (2) [1][1] (3) [1] (4) пустая строка (5) скрипт не будет выполнен, так как содержит ошибки</p>
23.	<p>Укажите результат выполнения скрипта:</p> <pre>tpl=1,2 class Foo (object): tpl+=(3,) ob = Foo() print (tpl, ob.tpl)</pre>	<p>(1) (1, 2, 3) (1, 2, 3) (2) (1, 2) (1, 2, 3) (3) (1, 2) (1, 2) (4) скрипт не будет выполнен, так как содержит ошибки</p>
24.	<p>Укажите результат выполнения скрипта:</p> <pre>lst=[0] class Foo (object): lst=lst+[1] ob = Foo() print (lst, ob.lst)</pre>	<p>(1) [0] [0] (2) [0] [0, 1] (3) [0, 1] [0, 1] (4) скрипт не будет выполнен, так как содержит ошибки</p>
25.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo (object): def obj (self): return [0] def __init__(self,*dt,**mp): self.obj=lambda i : 0 def __new__(cls,*dt,**mp):</pre>	<p>(1) <class 'list'> (2) <class 'int'> (3) <class 'str'> (4) скрипт не будет выполнен, т.к. код содержит ошибки</p>



	<pre>cls.obj=lambda i : "0" return object.__new__(cls,*dt,**mp) ob = Foo() print (type(ob.obj(0)))</pre>	
26.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo (object): def obj (self): return 0 obj = '0' ob = Foo() print (type(ob.obj()))</pre>	<p>(1) <class 'int'> (2) <class 'str'> (3) <class 'method'> (4) скрипт не будет выполнен, так как содержит ошибки</p>
27.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo (object): def value(self, x): return x.str def value(self, x): pass def value(self, x): return x ob = Foo() print (type(ob.value(0)))</pre>	<p>(1) <class 'int'> (2) <class 'str'> (3) <class 'NoneType'> (4) скрипт не будет выполнен, так как содержит ошибки</p>
28.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo (object): obj=0 def obj(self): return 'x' ob = Foo() print (type(ob.obj)) print (type(ob.obj()))</pre>	<p>(1) <class 'int'> <class 'int'> (2) <class 'int'> <class 'NoneType'> (3) <class 'method'> <class 'str'> (4) скрипт не будет выполнен, так как содержит ошибки</p>
29.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo (object): def value(self, x): return [x] def __init__(self): self.value = lambda i: {'0':i} def value(self, x): return (x,) ob = Foo() print (ob.value(0))</pre>	<p>(1) [0] (2) (0,) (3) {'0': 0} (4) скрипт не будет выполнен, так как содержит ошибки</p>
30.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo (object): def __new__(cls,*dt,**mp): cls.obj=lambda i : str(i) return object.__new__(cls,*dt,**mp) def obj(self, x): pass ob = Foo() print (type(ob.obj()))</pre>	<p>(1) <class 'str'> (2) <class 'str' 'NoneType'> (3) <class 'str' 'method'> (4) скрипт не будет выполнен, так как содержит ошибки</p>
31.	<p>В языке Python доступ через instance_ class_ attribute:</p>	<p>(1) разрешен к любым атрибутам (2) разрешен к любым атрибутам, кроме</p>



		помеченных специальными идентификаторами (3) разрешен к любым атрибутам, кроме атрибутов со специальными именами (4) запрещен ко всем атрибутам, кроме атрибутов со специальными именами
32.	В языке Python доступ через <code>instance.attribute</code> разрешен к любым атрибутам кроме:	(1) атрибутов с идентификаторами <code>private</code> (2) атрибутов, имена которых начинаются и кончаются на подчеркивание (3) атрибутов, имена которых начинаются и кончаются на двойное подчеркивание (4) атрибутов, имена которых начинаются на двойное подчеркивание и кончаются не на подчеркивание
33.	В языке Python доступ через <code>instance._class__attribute</code> разрешен к:	(1) атрибутам, имена которых начинаются и кончаются на подчеркивание (2) атрибутам, имена которых начинаются и кончаются на двойное подчеркивание (3) атрибутам, имена которых начинаются на подчеркивание и кончаются не на подчеркивание (4) атрибутам, имена которых начинаются на двойное подчеркивание и кончаются не на подчеркивание
34.	В языке Python инкапсуляция достигается:	(1) путем введения градаций доступности данных и методов класса, обязательных к использованию (2) путем четкого разделения данных и методов класса на закрытые и открытые средствами языка (3) путем соглашения между программистами об условном обозначении закрытых и открытых данных и полей (4) никак не достигается
35.	В языке Python прямой доступ:	(1) разрешен к любым атрибутам (2) разрешен к любым атрибутам, кроме помеченных специальными идентификаторами (3) разрешен к любым атрибутам, кроме атрибутов со специальными именами (4) запрещен ко всем атрибутам, кроме атрибутов со специальными именами
36.	В языке Python доступ через <code>instance.attribute</code> :	(1) разрешен к любым атрибутам (2) разрешен к любым атрибутам, кроме помеченных специальными идентификаторами (3) разрешен к любым атрибутам, кроме атрибутов со специальными именами (4) запрещен ко всем атрибутам, кроме атрибутов со специальными именами
37.	Встроенный метод <code>__setattr__</code> вызывается:	(1) автоматически, при попытке присвоить значение атрибута через <code>instance.attribute</code> (2) автоматически, при попытке присвоить значение атрибута через <code>instance.attribute</code> , если не найден атрибут, к которому идет обращение (3) автоматически, при попытке присвоить значение атрибута через <code>instance.attribute</code> или <code>instance._class__attribute</code> (4) автоматически, при попытке присвоить значение атрибута через <code>instance.attribute</code> или <code>instance._class__attribute</code> , если не найден атрибут, к которому идет обращение



		(5) только явно
38.	Встроенный метод <code>__delattr__</code> вызывается:	(1) автоматически, при попытке удалить атрибут через <code>instance.attribute</code> (2) автоматически, при попытке удалить атрибут через <code>instance.attribute</code> , если не найден атрибут, к которому идет обращение (3) автоматически, при попытке удалить атрибут через <code>instance.attribute</code> или <code>instance._class__attribute</code> (4) автоматически, при попытке удалить атрибут через <code>instance.attribute</code> или <code>instance._class__attribute</code> , если не найден атрибут, к которому идет обращение (5) только явно
39.	Одиночное подчеркивание в начале имени атрибута класса указывает на:	(1) то, что атрибут является свойством (2) то, что он приватный (3) то, что он приватный и доступ к нему не может быть получен через <code>instance._attribute</code> (4) то, что атрибут является атрибутом класса, т.е. к нему можно получить доступ без инстанцирования класса (5) в случае атрибутов класса одиночное подчеркивание ничего не означает
40.	Прямой доступ к атрибуту класса нельзя получить, если:	(1) перед определением атрибутом стоит идентификатор <code>private</code> (2) если имя атрибута начинается с подчеркивания и кончается на подчеркивание (3) если имя атрибута начинается с двойного подчеркивания и кончается на двойное подчеркивание (4) в языке Python можно получить прямой доступ к любому атрибуту
41.	В языке Python встроенный метод <code>property()</code> используется для:	(1) получения информации об объекте, метод которого вызывается (2) получения информации обо всех свойствах объекта, метод которого вызывается (3) реализации доступа к определенному атрибуту класса как к свойству (4) реализации доступа к любым атрибутам класса как к свойствам
42.	Встроенный метод <code>__getattr__</code> вызывается:	(1) при попытке получить значение атрибута через <code>instance.attribute</code> (2) при попытке получить значение атрибута через <code>instance.attribute</code> , если не найден атрибут, к которому идет обращение (3) при попытке получить значение атрибута через <code>instance.attribute</code> или <code>instance._class__attribute</code> (4) при попытке получить значение атрибута через <code>instance.attribute</code> или <code>instance._class__attribute</code> , если не найден атрибут, к которому идет обращение
43.	Укажите результат выполнения скрипта: <pre>class Foo (object): def method (self): print ('1', end=' ') def _method_ (self): print ('2', end=' ') def __method__ (self): print ('3', end=' ')</pre>	(1) 1 2 3 (2) 1 2 и сообщение об ошибке (3) 1 и сообщение об ошибке (4) сообщение об ошибке



	<pre>o=Foo() o.method() o._method_() o.__method__()</pre>	
44.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo (object): x=1 __x=2 x__=3 print (Foo.x) print (Foo.__x) print (Foo.x__)</pre>	<p>(1) 1 2 3 (2) 1 2 и сообщение об ошибке (3) 1 и сообщение об ошибке (4) сообщение об ошибке</p>
45.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo (object): def method (self): print ('1', end=' ') def __method (self): print ('2', end=' ') def method__ (self): print ('3', end=' ') o=Foo() o.method() o.__method() o.method__()</pre>	<p>(1) 1 2 3 (2) 1 2 и сообщение об ошибке (3) 1 и сообщение об ошибке (4) сообщение об ошибке</p>
46.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo (object): x=1 __x=2 x__=3 print (Foo.x+Foo.__x) print (Foo.__x+Foo.x__)</pre>	<p>(1) 3 5 (2) 3 3 (3) 3 Ошибка в шестой строке (4) Ошибка в пятой строке</p>
47.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo (object): def method (self): print (1) def __method (self): print (2) def __method (self): print (3) o=Foo() o.method() o.__method() o.__method()</pre>	<p>(1) 1 2 3 (2) 1 2 и сообщение об ошибке (3) 1 и сообщение об ошибке (4) сообщение об ошибке</p>
48.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo (object): x=1 __x__=2 x__=3 print (Foo.x, Foo.__x__, Foo.__x__)</pre>	<p>(1) 1 2 3 (2) 1 2 и сообщение об ошибке (3) 1 и сообщение об ошибке (4) сообщение об ошибке</p>
49.	<p>Укажите результат выполнения скрипта:</p>	<p>(1) 1, b_0</p>



	<pre>class Foo(object): __value__=0 def __getattr__(self, name): return (name+' '+self.__value__._str__()) def __setattr__(self, name, value): object.__setattr__(self, '__value__', value) o = Foo() o.a=1 print (o.a, o.b)</pre>	<p>(2) 1, b_1 (3) a_0, b_0 (4) a_1, b_1 (5) скрипт будет работать бесконечно долгое время</p>
50.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo(object): __value__={} def __getattr__(self, name): if name in self.__value__: return self.__value__[name] else: return name+'_atr' def __setattr__(self, name, value): object.__setattr__(self, '__value__', value) o = Foo() o.a=1 print (o.a, o.b)</pre>	<p>(1) a_atr b_atr (2) 1 b_atr (3) 1 None (4) a_atr None (5) в скрипте содержится ошибка</p>
51.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo(object): __value__=0 def __getattr__(self, name): print (name+' '+self.__value__._str__(), end = ' '), def __setattr__(self, name, value): object.__setattr__(self, '__value__', value) o = Foo() o.a=1 print (o.a, o.b)</pre>	<p>(1) a_1 a_1 b_1 b_1 (2) a_1 a_0 b_1 b_0 (3) a_1 b_1 None None (4) a_1 b_1 a_0 b_0 (5) a_1 b_1 None None</p>
52.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo(object): __value__={} def __getattr__(self, name): if name in self.__value__: return self.__value__[name] else: return (name+'_atr') def __setattr__(self, name, value): self.__value__[name]=value o = Foo() o.a=12 print (o.a, o.b) print (o.__value__)</pre>	<p>(1) 12, 12 {'a': 12, 'b': 12} (2) 12, b_atr {'a': 12} (3) a_atr, b_atr {} (4) 12, b_atr ошибка при выполнении строки 'print o.__value__'; (5) ошибка при выполнении строки 'print o.a, o.b'</p>
53.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo(object): __value__=0 def __getattr__(self, name): return (name+' '+self.__value__._str__()) def __setattr__(self, name, value): self.__value__=value</pre>	<p>(1) 1, b_0 (2) 1, b_1 (3) a_0, b_0 (4) a_1, b_1 (5) будет выдано сообщение об ошибке</p>



	<pre>o = Foo() o.a=1 print (o.a, o.b)</pre>	
54.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo(object): __value__ = {} def __getattr__(self, name): if self.__value__.has_key(name): return self.__value__[name] else: return name+'_atr' def __setattr__(self, name, value): object.__setattr__(self, name, value) o = Foo() o.a=1 print (o.a, o.b)</pre>	<ul style="list-style-type: none">(1) a_atr b_atr(2) 1 b_atr(3) 1 None(4) a_atr None(5) скрипт будет работать бесконечно долгое время
55.	<p>Встроенный атрибут <code>__mro__</code> в языке Python:</p>	<ul style="list-style-type: none">(1) возвращает порядок разрешения методов и позволяет его изменять(2) возвращает порядок разрешения методов, но не позволяет его изменять(3) возвращает список доступных методов и атрибутов класса(4) возвращает список доступных методов класса
56.	<p>При наследовании в языке Python порядок разрешения методов для "новых" классов (т.е. классов, наследников класса <code>object</code>) определяется следующим образом:</p>	<ul style="list-style-type: none">(1) рассматривается первый суперкласс и далее его суперкласс, если метод не найден, то рассматривается второй суперкласс(2) рассматривается первый суперкласс, если метод не найден, то рассматривается второй суперкласс и т.д., если во всех суперклассах метод не найден, то рассматривается суперклассы первого суперкласса и т.д.(3) рассматривается последний суперкласс и далее его суперкласс, если метод не найден, то рассматривается предпоследний суперкласс(4) рассматривается последний суперкласс, если метод не найден, то рассматривается предпоследний суперкласс и т.д., если во всех суперклассах метод не найден, то рассматривается суперклассы последнего суперкласса и т.д.
57.	<p>При наследовании в языке Python порядок разрешения методов для "классических" классов (т.е. классов, не являющихся наследниками класса <code>object</code>) определяется следующим образом:</p>	<ul style="list-style-type: none">(1) рассматривается первый суперкласс и далее его суперклассы, если метод не найден, то рассматривается второй суперкласс(2) рассматривается первый суперкласс, если метод не найден, то рассматривается второй суперкласс и т.д., если во всех суперклассах метод не найден, то рассматривается суперклассы первого суперкласса и т.д.(3) рассматривается последний суперкласс и далее его суперклассы, если метод не найден, то рассматривается предпоследний суперкласс(4) рассматривается последний суперкласс, если метод не найден, то рассматривается предпоследний суперкласс и т.д., если во всех суперклассах метод не найден, то рассматривается суперклассы последнего суперкласса и т.д.



58.	При наследовании в языке Python:	(1) подклассы наследуют все атрибуты суперкласса (2) подклассы наследуют все атрибуты суперкласса, кроме специально помеченных (3) подклассы наследуют все атрибуты суперкласса, кроме приватных (4) подклассы не наследуют никакие атрибуты суперкласса, кроме специально помеченных
59.	При наследовании в языке Python приватные методы:	(1) не наследуются (2) наследуются, если помечены специальным идентификатором (3) наследуются и доступны как через instance._child_method(), так и через instance._parent_method() (child – имя подкласса, parent – имя суперкласса) (4) наследуются и доступны только через instance._parent_method()
60.	Укажите результат выполнения скрипта: class Foo (object): __value__=12 def __truediv__ (self, x): return self.__value__ /x*2 class Bar(Foo): pass o = Bar() o=o/3 o=o/2 print (o)	(1) 2.0 (2) 4.0 (3) 8.0 (4) скрипт не будет выполнен, т.к. код содержит ошибки
61.	Укажите результат выполнения скрипта: class Foo (object): __value__=[] def append(self): self.__value__ +=[1] def __append(self): self.__value__ +=[2] class Bar(Foo): def append(self): self.__value__ +=[3] o = Bar() o.append() o.__append() print (o.__value__)	(1) [1, 2] (2) [1, 3] (3) [3, 2] (4) [3, 1] (5) скрипт не будет выполнен, т.к. код содержит ошибки
62.	Укажите результат выполнения скрипта: class Foo (object): __value__=0 def __add__ (self, x): ob = Foo() ob.__value__ =self.__value__ +x*2 return ob def __repr__(self): return str(self.__value__) class Bar(Foo): def __repr__(self): return str(self.__value__ +1) o = Bar() o=o+3	(1) 3 (2) 4 (3) 6 (4) 7 (5) скрипт не будет выполнен, т.к. код содержит ошибки



	print (o)	
63.	Укажите результат выполнения скрипта: class Foo (object): def __method__ (self): print ('1') def method (self): print ('2') class Bar (Foo): def method (self): print ('3') o = Bar() o.method() o.__method__()	(1) 1 1 (2) 3 3 (3) 2 1 (4) 3 1 (5) скрипт не будет выполнен, т.к. код содержит ошибки
64.	Укажите результат выполнения скрипта: class Foo (object): __value__ =0 def __add__ (self, x): return self.__value__ +x*2 class Bar(Foo): pass o = Bar() o=o+1 o=o+2 print (o)	(1) 3 (2) 4 (3) 5 (4) 6 (5) скрипт не будет выполнен, т.к. код содержит ошибки
65.	Укажите результат выполнения скрипта: class Foo (object): __value__ =0 def add(self): self.__value__ +=1 def __add(self): self.__value__ +=2 class Bar(Foo): def add(self): self.__value__ +=3 o = Bar() o.add() o.__add() print (o.__value__)	(1) 3 (2) 4 (3) 5 (4) 6 (5) скрипт не будет выполнен, т.к. код содержит ошибки
66.	Укажите результат выполнения скрипта: class Foo (object): __val__ =0 def m1 (self): self.__val__ +=1 def m2 (self): self.__val__ +=2 class Bar (Foo): __val__ =1 def m2 (self): self.__val__ -=1 def m3 (self): self.__val__ -=2 class Baz (Bar): __val__ =2 def m1 (self): self.__val__ +=5 o = Baz() o.m1() o.m2()	(1) 1 (2) 2 (3) 3 (4) 4 (5) скрипт не будет выполнен, т.к. код содержит ошибки



	<pre>o.m3() print (o.__val__)</pre>	
67.	<pre>Укажите результат выполнения скрипта: class Foo (object): def __init__(self, value): self.__val__=value def __sub__(self, x): return Foo(self.__val__-x) def __add__(self, x): return Foo(self.__val__+x) class Bar (Foo): def __sub__(self, x): return Bar(self.__val__-x*3) def __mul__(self, x): return Bar(self.__val__*x*3) class Baz (Bar): def __mul__(self, x): return Baz(self.__val__*x*2) o = Baz(1) o+=3 o*=2 o-=1 print (o.__val__)</pre>	<p>(1) 21 (2) 13 (3) 23 (4) 15 (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>
68.	<pre>Укажите результат выполнения скрипта: class Foo (object): __val__=[0] def m1 (self): self.__val__+=[1] def m2 (self): self.__val__+=[2] class Bar (Foo): __val__=[-1] def m1 (self): self.__val__+=[3] def m3 (self): self.__val__+=[4] class Baz (Bar): __val__=[-2] def m2 (self): self.__val__+=[5] o = Baz() o.m1() o.m2() o.m3() print (o.__val__)</pre>	<p>(1) [0, 1, 2, 4] (2) [-1, 3, 5, 4] (3) [-2, 1, 2, 4] (4) [-2, 3, 5, 4] (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>
69.	<pre>Укажите результат выполнения скрипта: class Foo (object): def __init__(self, value): self.__val__=value def __sub__(self, x): return Foo(self.__val__-x) def __add__(self, x): return Foo(self.__val__+x) class Bar (Foo): def __sub__(self, x): return Bar(self.__val__-x*2) def __mul__(self, x): return Bar(self.__val__*x*2) class Baz (Bar): def __mul__(self, x): return Baz(self.__val__*x) o = Baz(1) o*=2 o-=4 o+=3 print (o.__val__)</pre>	<p>(1) 1 (2) 2 (3) -3 (4) 0 (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>
70.	<pre>Укажите результат выполнения скрипта: class Foo (object): def method1 (self): print ('1') def method2 (self): print ('2')</pre>	<p>(1) 1 2 3+ (2) 1 2+ 3++ (3) 1 2+ 3+ (4) 1 2 3++</p>



	<pre>class Bar (Foo): def method2 (self): print ('2+') def method3 (self): print ('3+') class Baz (Bar): def method3 (self): print ('3++') o = Baz() o.method1() o.method2() o.method3()</pre>	(5) скрипт не будет выполнен, т.к. код содержит ошибки
71.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo (object): def __init__ (self, value): self.__val__=value def __sub__ (self, x): return Foo(self.__val__-x) def __add__ (self, x): return Foo(self.__val__+x) class Bar (Foo): def __sub__ (self, x): return Bar(self.__val__-x*3) def __mul__ (self, x): return Bar(self.__val__*x*3) class Baz (Bar): def __sub__ (self, x): return Baz(self.__val__-x*2) o = Baz(1) o*=2 o-=1 o+=3 print (o.__val__)</pre>	(1) 3 (2) 5 (3) 6 (4) 8 (5) скрипт не будет выполнен, т.к. код содержит ошибки
72.	<p>Скрипт содержит следующий код:</p> <pre>class Class1 (object): pass class Class2 (Class1): pass class Class3 (_____): pass</pre> <p>Среди приведенных ниже фрагментов укажите все варианты кода, при подстановке которого вместо знаков подчеркивания скрипт будет корректным:</p>	(1) object, Class1, Class2 (2) object, Class2, Class1 (3) Class1, object, Class2 (4) Class2, object, Class1 (5) Class2, Class1, object (6) Class2, object
73.	<p>Скрипт содержит следующий код:</p> <pre>class Foo (object): pass class Bar (object): pass class Baz (_____): pass</pre> <p>Среди приведенных ниже фрагментов укажите все варианты кода, при подстановке которого вместо знаков подчеркивания скрипт будет корректным:</p>	(1) object, Foo (2) Foo, object (3) object, Bar (4) Bar, Foo (5) Foo, Bar, object
74.	<p>Скрипт содержит следующий код:</p> <pre>class Foo (object): pass class Bar (Foo): pass class Baz (_____): pass</pre> <p>Среди приведенных ниже фрагментов укажите все варианты кода, при подстановке которого вместо</p>	(1) object, Foo (2) Foo, object (3) Foo, Bar (4) Bar, Foo (5) Foo, object, Bar (6) Bar, object



	знаков подчеркивания скрипт будет корректным:	
75.	Скрипт содержит следующий код: class Foo (object): class Class1 (object): pass class Class2 (object): pass class Class3 (____): pass Среди приведенных ниже фрагментов укажите все варианты кода, при подстановке которого вместо знаков подчеркивания скрипт будет корректным:	(1) object, Class1 (2) object, Class2 (3) Class1, object (4) Class2, object (5) Class1, Class2
76.	Скрипт содержит следующий код: class Foo (object): pass class Bar (Foo): pass class Baz (____): pass Среди приведенных ниже фрагментов укажите вариант кода, при подстановке которого вместо знаков подчеркивания скрипт будет корректным:	(1) object, Foo, Bar (2) Foo, object, Bar (3) Foo, Bar, object (4) Bar, Foo, object (5) Foo, object, Bar (6) object, Bar
77.	Укажите результат выполнения скрипта: class Foo: def method1 (self): return 1 class Bar (Foo): pass class Baz (Bar): def method1 (self): return 2 class Lo (Baz, Bar): pass o = Lo() print (o.method1())	(1) пустая строка (2) 1 (3) 2 (4) скрипт не будет выполнен, т.к. код содержит ошибки
78.	Укажите результат выполнения скрипта: class Foo(object): def method1 (self): return 1 class Bar (Foo): pass class Baz (Foo): def method1 (self): return 2 class Lo (Bar, Foo, Baz): pass o = Lo() print (o.method1())	(1) пустая строка (2) 1 (3) 2 (4) скрипт не будет выполнен, т.к. код содержит ошибки
79.	Укажите результат выполнения скрипта: class Foo: def method1 (self): return 1	(1) пустая строка (2) 1 (3) 2 (4) скрипт не будет выполнен, т.к. код содержит



	<pre>class Bar (Foo): pass class Baz (Foo): def method1 (self): return 2 class Lo (Bar, Foo): pass o = Lo() print (o.method1())</pre>	ошибки
80.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo (object): def method1 (self): return 1 class Bar (Foo): pass class Baz (Foo): def method1 (self): return 2 class Lo (Bar, Baz): pass o = Lo() print (o.method1())</pre>	(1) пустая строка (2) 1 (3) 2 (4) 3
81.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo: def method1 (self): return 1 class Bar (Foo): pass class Baz (Foo): def method1 (self): return 2 class Lo (Bar, Baz): pass o = Lo() print (o.method1())</pre>	(1) пустая строка (2) 1 (3) 2 (4) скрипт не будет выполнен, т.к. код содержит ошибки
82.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo(object): def method1 (self): return 1 class Bar (Foo): pass class Baz (Bar): def method1 (self): return 2 class Lo (Bar, Baz): pass o = Lo() print (o.method1())</pre>	(1) пустая строка (2) 1 (3) 2 (4) скрипт не будет выполнен, т.к. код содержит ошибки
83.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo(object): def __setattr__(self, name, value): object.__setattr__(self, name+'1', value) class Bar (Foo):</pre>	(1) 0 (2) 1 (3) 2 (4) a1 (5) скрипт не будет выполнен, т.к. код содержит



	<pre>a=0 o = Bar() o.a=1 print (o.a)</pre>	ошибки
84.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo(object): a=1 def __getattr__(self, name): return (str(self.a)+name) class Bar (Foo): a=0 o = Bar() o.a=2 print (o.a, o.b)</pre>	<p>(1) 0 0b (2) 2 1b (3) 1 1b (4) 2 2b (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>
85.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo(object): a=1 def __setattr__(self, name, value): object.__setattr__(self, name, value*2) class Bar (Foo): a=0 o = Bar() o.a=-1 print (o.a)</pre>	<p>(1) -2 (2) -1 (3) 0 (4) 2 (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>
86.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo(object): __value__ = {} def __getattr__(self, name): if self.__value__.has_key(name): return self.__value__[name] else: return (name+'_atr') def __setattr__(self, name, value): self.__value__[name]=value+1 class Bar (Foo): a=0 o = Bar() o.a=5 print (o.a)</pre>	<p>(1) 0 (2) 1 (3) 5 (4) 6 (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>
87.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo(object): def __setattr__(self, name, value): object.__setattr__(self, name, value+1) class Bar (Foo): a=0 o = Bar() o.a=2 print (o.a)</pre>	<p>(1) 0 (2) 1 (3) 2 (4) 3 (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>
88.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo(object): __value__ =1 def __getattr__(self, name):</pre>	<p>(1) 0 (2) 3 (3) 0_a (4) 3_a</p>



	<pre>return (str(self.__value__)+name) class Bar (Foo): a=0 o = Bar() o.a=3 print (o.a)</pre>	(5) скрипт не будет выполнен, т.к. код содержит ошибки
89.	Для того, чтобы функция, определенная для типа А работала с типом В необходимо:	(1) чтобы тип В был наследником типа А (2) для типа В были определены все методы класса А (3) для типа В были определены все методы класса А, применяющиеся в функции (4) в языке Python функции работают только с теми типами, для которых определены
90.	В языке Python метод является абстрактным, если:	(1) перед его определением стоит ключевое слово abstract; (2) его имя начинается с двойного подчеркивания (3) если он помечен как @abstractmethod (4) в языке Python не существует встроенной программной реализации абстрактных методов
91.	Исключение NotImplementedError используется для:	(1) обозначения, что метод, в котором оно сгенерировано, не поддерживает тип переданного аргумента (2) обозначения, что некоторый функционал не реализован (3) обозначения, что метод, в котором оно сгенерировано, не может быть переопределен в наследниках (4) в языке Python нет встроенного исключения NotImplementedError
92.	Для того, чтобы функция, определенная для типа А работала с типом В необходимо:	(1) чтобы тип В был наследником типа А (2) для типа В были определены все методы класса А (3) для типа В были определены все методы класса А, применяющиеся в функции (4) в языке Python функции работают только с теми типами, для которых определены
93.	Проверка на совместимость типов производится с помощью оператора	(1) is (2) isinstance (3) isinstanceof (4) в Python не существует подобного механизма
94.	Положительный результат выполнения проверки на совместимость типов оператором isinstance означает, что	(1) переменная относится к указанному классу (2) переменная относится к классу, потомком которого является указанный (3) переменная либо относится к указанному классу, либо относится к классу, являющемуся потомком указанного (4) в Python не существует оператора isinstance
95.	Укажите результат выполнения скрипта: <pre>from abc import abstractmethod class Base(): __val__=0 def __add__(self, x): return self.__val__+x.__val__ @abstractmethod def fun (self):</pre>	(1) 0 (2) 1 (3) 2 (4) 2 и сообщение о том, что у объекта нет метода fun (5) скрипт выдаст исключение NotImplementedError



	<pre>pass class Foo(Base): __val__ = 1 def fun(self): return 2 o=Foo() b=Foo() print ((b+o).fun())</pre>	
96.	<p>Укажите результат выполнения скрипта:</p> <pre>from abc import abstractmethod, ABCMeta class Base(metaclass=ABCMeta): def __init__(self, x): self.__val__ = x def __add__(self, x): return Base(self.__val__ + self.__val__) @abstractmethod def fun(self): pass def val(self): return self.__val__ class Foo(Base): def fun(self): return self.__val__ o=Foo(1) b=Foo(2) print ((b+o.fun()).val())</pre>	<p>(1) 2 (2) 3 (3) 4 (4) скрипт не будет выполнен, так как у объекта не будет метода fun (5) скрипт выдаст исключение TypeError с сообщением о невозможности инстанцирования абстрактного класса</p>
97.	<p>Укажите результат выполнения скрипта:</p> <pre>from abc import abstractmethod, ABCMeta class Base(metaclass=ABCMeta): def __init__(self, x): self.__val__ = x def __add__(self, x): return Base(self.__val__ + self.__val__) @abstractmethod def fun(self): pass def val(self): return self.__val__ class Foo(Base): __val__ = 0 def __add__(self, x): return Foo(self.__val__ + self.__val__) o=Foo(1) b=Foo(2) print (b+o.fun())</pre>	<p>(1) 2 (2) 3 (3) 4 (4) скрипт выдаст исключение TypeError с сообщением о невозможности инстанцирования абстрактного класса (5) скрипт не будет выполнен, так как содержит ошибки</p>
98.	<p>Укажите результат выполнения скрипта:</p> <pre>class base(object): def __init__(self): raise NotImplementedError def fun(self): return 1 class Foo(base): def fun2(self): return 2</pre>	<p>(1) 2 (2) 3 (3) 4 (4) скрипт не будет выполнен, так как у объекта не будет метода fun (5) скрипт выдаст исключение NotImplementedError</p>



	<pre>o=Foo() print (o.fun()+o.fun2())</pre>	
99.	<p>Укажите результат выполнения скрипта:</p> <pre>class base(object): def __add__(self, x): raise NotImplementedError def fun (self): return 1 class Foo(base): def fun(self): return 2 o=Foo() print (o+o.fun())</pre>	<p>(1) 2 (2) 3 (3) 4 (4) скрипт не будет выполнен, так как у объекта не будет метода fun (5) скрипт выдаст исключение NotImplementedError</p>
100.	<p>Укажите результат выполнения скрипта:</p> <pre>class base(object): def __add__(self, x): raise NotImplementedError def fun (self): return 1 class Foo(base): def fun(self): return 2 o=Foo() print (o.fun()+o.fun())</pre>	<p>(1) 2 (2) 3 (3) 4 (4) скрипт не будет выполнен, так как у объекта не будет метода fun (5) скрипт выдаст исключение NotImplementedError</p>
101.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo(object): val = 0 def fun(self): try: self.val[0]=0 return self.val except TypeError: return self.val o=Foo() o.val=(1,2,3) print (o.fun(), end=' ') o.val='123' print (o.fun(), end=' ')</pre>	<p>(1) (0, 2, 3) 023 (2) (1, 2, 3) 023 (3) (0, 2, 3) 123 (4) (1, 2, 3) 123 (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>
102.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo(object): val = 0 def fun(self): try: self.val[0]=0 return self.val except TypeError: return self.val o=Foo() o.val=[1,2,3] print (o.fun(), end=' ') o.val=(1, 2, 3) print (o.fun(), end=' ')</pre>	<p>(1) [0, 2, 3] (0, 2, 3) (2) [0, 2, 3] (1, 2, 3) (3) [1, 2, 3] (0, 2, 3) (4) [1, 2, 3] (1, 2, 3) (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>



103.	Укажите результат выполнения скрипта: <pre>class Foo(object): val = 0 def fun(self): try: self.val[0]+=self.val[0] return self.val except TypeError: return self.val o=Foo() o.val=12 print (o.fun(), end=' ') o.val='12' print (o.fun(), end=' ')</pre>	(1) 12 12 (2) 12 112 (3) 22 12 (4) 22 112 (5) скрипт не будет выполнен, т.к. код содержит ошибки
104.	Укажите результат выполнения скрипта: <pre>class Foo(object): val = 0 def fun(self): try: return self.val[-1] except TypeError: return self.val o=Foo() o.val='Hello' print (o.fun()) o.val=15 print (o.fun())</pre>	(1) 0 0 (2) o o (3) o 5 (4) o 15 (5) скрипт не будет выполнен, т.к. код содержит ошибки
105.	Укажите результат выполнения скрипта: <pre>class Foo(object): val = 0 def fun(self): try: return self.val[-1] except TypeError: return self.val o=Foo() o.val=(1,2,3,4) print (o.fun()) o.val={'0':1,'1':2,'2':3,'3':4} print (o.fun())</pre>	(1) 4 4 (2) 4 {'0': 1, '1': 2, '2': 3, '3': 4} (3) 4 и сообщение об ошибке доступа по ключу (4) (1, 2, 3, 4) {'0': 1, '1': 2, '2': 3, '3': 4} (5) скрипт не будет выполнен, т.к. код содержит ошибки
106.	Укажите результат выполнения скрипта: <pre>class Foo(object): val = 0 def fun(self): try: return self.val[-1] except TypeError: return self.val o=Foo() o.val=[1,2,3,4] print o.fun(), o.val='1234' print (o.fun())</pre>	(1) 4 4 (2) 4 1234 (3) [1, 2, 3, 4] 4 (4) [1, 2, 3, 4] 1234 (5) скрипт не будет выполнен, т.к. код содержит ошибки



107.	Укажите результат выполнения скрипта: <pre>class Foo(object): def meth (self, x=1): return x*2 def meth (self, *x): s=0 for i in x: s+=i return s o=Foo() print (o.meth(-1)+o.meth(2))</pre>	(1) 0 (2) 1 (3) 2 (4) 3 (5) скрипт не будет выполнен, так как содержит ошибки
108.	Укажите результат выполнения скрипта: <pre>class Foo(object): def meth (self, x): return x*2 def meth (self, x, *y): s=x for i in y: s+=i*2 return s o=Foo() print (o.meth(3)+o.meth(4, 5))</pre>	(1) 14 (2) 20 (3) 17 (4) 11 (5) скрипт не будет выполнен, так как содержит ошибки
109.	Укажите результат выполнения скрипта: <pre>class Foo(object): def meth (self, x): return x*2 def meth (self, x, y=-2): return x+y o=Foo() print (o.meth(3)+o.meth(4))</pre>	(1) 3 (2) 8 (3) 9 (4) 14 (5) скрипт не будет выполнен, так как содержит ошибки
110.	Укажите результат выполнения скрипта: <pre>class Foo(object): def meth (self, x): return x*2 def meth (self, x, y=3): return (x+y)*3 o=Foo() print (o.meth(1)+o.meth(1, 2))</pre>	(1) 4 (2) 11 (3) 14 (4) 21 (5) скрипт не будет выполнен, так как содержит ошибки
111.	Укажите результат выполнения скрипта: <pre>class Foo(object): def meth (self, x=0): return x def meth (self, x, y=2): return (x+y)/2 o=Foo() print (o.meth()+o.meth(2))</pre>	(1) 0 (2) 1 (3) 2 (4) 3 (5) скрипт не будет выполнен, так как содержит ошибки
112.	Укажите результат выполнения скрипта: <pre>class Foo(object): def meth (self, x=0): return x*2 def meth (self, *x): s=0 for i in x: s+=i return s</pre>	(1) 4 (2) 3 (3) 2 (4) 1 (5) скрипт не будет выполнен, так как содержит ошибки



	<pre>o=Foo() print (o.meth(-1)+o.meth(2,3))</pre>	
113.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo(str): def __init__(self, x): self.__val__=x def __add__(self, val): return Foo(int(self.__val__)+int(val.__val__)) def __str__(self): return str(self.__val__) print ((Foo('1')+Foo('2'))*2)</pre>	<p>(1) 1212 (2) 33 (3) 24 (4) 6 (5) скрипт не будет выполнен, так как содержит ошибки</p>
114.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo(int): def __init__(self, x): self.__val__=x def __add__(self, val): return Foo(str(self.__val__)+str(val.__val__)) def __str__(self): return str(self.__val__) print ((Foo('1')+Foo('2'))*2)</pre>	<p>(1) 1212 (2) 33 (3) 24 (4) 6 (5) скрипт не будет выполнен, так как содержит ошибки</p>
115.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo(float): def __init__(self, x): self.__val__=x def __add__(self, val): return Foo(int(self.__val__)+int(val.__val__)) def __str__(self): return str(self.__val__) print ((Foo('1')+Foo('2'))*2)</pre>	<p>(1) 6 (2) 6.0 (3) 24 (4) 24.0 (5) скрипт не будет выполнен, так как содержит ошибки</p>
116.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo(object): def __init__(self, x): self.__val__=x def __add__(self, val): return Foo(self.__val__+val.__val__) def __str__(self): return str(self.__val__) print (Foo(1)+Foo(2), Foo('1')+Foo('2'))</pre>	<p>(1) 3 3 (2) 3 12 (3) 12 3 (4) 12 12 (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>
117.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo(int): def __init__(self, x): self.__val__=x def __add__(self, val): return Foo(val.__val__.__class__(self.__val__)+val.__val__) def __str__(self): return str(self.__val__) print (Foo(1)+Foo('2'), Foo('1')+Foo(2))</pre>	<p>(1) 3 3 (2) 3 12 (3) 12 3 (4) 12 12 (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>
118.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo(int):</pre>	<p>(1) 0 (2) 1</p>



	<pre>def __init__(self, x): self.__val__=x def __str__(self): return str(self.__val__) print (Foo('12')%Foo('2'))</pre>	(3) 2 (4) 122 (5) скрипт не будет выполнен, т.к. код содержит ошибки
119.	Укажите результат выполнения скрипта: <pre>def fun1(f): print (f(1), end=' ') @fun1 @fun1 def m(x): return x+2</pre>	(1) пустая строка (2) 1 (3) 3 (4) 5 (5) 3 и сообщение об ошибке (6) 5 и сообщение об ошибке
120.	Укажите результат выполнения скрипта: <pre>def fun1(f): return lambda i: f(i)+2 def fun2(f): print (f(3), end=' ') @fun1 @fun2 def m(x): return x+2</pre>	(1) пустая строка (2) 5 (3) 7 (4) 9 (5) скрипт не будет выполнен, так как содержит ошибки
121.	Укажите результат выполнения скрипта: <pre>def fun1(f): print (f(3), end=' ') def fun2(f): return lambda i: f(i)+2 @fun1 @fun2 def m(x): return x+2</pre>	(1) пустая строка (2) 5 (3) 7 (4) 9 (5) скрипт не будет выполнен, так как содержит ошибки
122.	Укажите результат выполнения скрипта: <pre>def fun1(f): print (f(1)) @fun1 def m(x): return x+1 @fun1 def m2(x): return x+2</pre>	(1) пустая строка (2) 1 1 (3) 2 3 (4) скрипт не будет выполнен, т.к. код содержит ошибки
123.	Укажите результат выполнения скрипта: <pre>def fun1(f): print (f(1)) def fun2(f): print (f(2)) @fun2 def m(x): return x+1 @fun1 def m(x): return x+2</pre>	(1) 1 2 (2) 2 3 (3) 2 2 (4) 3 3 (5) скрипт не будет выполнен, т.к. код содержит ошибки



124.	Укажите результат выполнения скрипта: <pre>def fun1(f): print (f(1)) @fun1 def m(x): return (x,) @fun1 def m2(x): return [x]</pre>	(1) пустая строка (2) (1,) [1] (3) [1] (1,) [1] (4) 1 1 (5) скрипт не будет выполнен, т.к. код содержит ошибки
125.	Укажите результат выполнения скрипта: <pre>class foo(object): def method (self): return '012' val = [method] x = foo() s = x.val del x print (s())</pre>	(1) method (2) 012 (3) None (4) сгенерируется исключение NameError (5) скрипт не будет выполнен, т.к. код содержит ошибки
126.	Укажите результат выполнения скрипта: <pre>class foo(object): val = (1,2,3) x = {'0':foo()} s = x['0'].val del x print (s)</pre>	(1) 0 (2) 1 (3) (1, 2, 3) (4) сгенерируется исключение NameError (5) скрипт не будет выполнен, т.к. код содержит ошибки
127.	Укажите результат выполнения скрипта: <pre>class foo(object): def method (): return '012' val = [method] x = foo() s = x.val[0] del x print (s())</pre>	(1) method (2) 012 (3) None (4) сгенерируется исключение NameError (5) скрипт не будет выполнен, т.к. код содержит ошибки
128.	Укажите результат выполнения скрипта: <pre>class foo(object): def __repr__(self): return 'class' x = [foo()] s = x[0] del x print (s)</pre>	(1) пустая строка (2) None (3) class (4) сгенерируется исключение NameError (5) скрипт не будет выполнен, т.к. код содержит ошибки
129.	Укажите результат выполнения скрипта: <pre>class foo(object): val = [1,2,3] x = foo() s = x.val[0] del x print (s)</pre>	(1) [1,2,3] (2) 1 (3) None (4) сгенерируется исключение NameError (5) скрипт не будет выполнен, т.к. код содержит ошибки
130.	Укажите результат выполнения скрипта:	(1) пустая строка



	<pre>class foo(object): def method(self): return 'call method' x = (foo().method,) s = x[0] del x print (s())</pre>	<p>(2) None (3) call method (4) сгенерируется исключение NameError (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>
131.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo: def __init__(self, lst): self.lst = lst def __iter__(self): return self def __next__(self): raise StopIteration for i in Foo(range(4)): print (i+1, end=' ')</pre>	<p>(1) пустая строка (2) 1 2 3 4 (3) 4 3 2 1 (4) скрипт выдаст исключение StopIteration (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>
132.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo: def __init__(self, lst): self.lst = lst def __iter__(self): return self def __next__(self): if self.lst: return self.lst.pop() else: raise StopIteration for i in Foo(list(range(4))): print (i+1, end=' ')</pre>	<p>(1) пустая строка (2) 1 2 3 4 (3) 4 3 2 1 (4) скрипт выдаст исключение StopIteration (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>
133.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo: def __init__(self, lst): self.lst = lst def __iter__(self): return self def __next__(self): if self.lst: return self.lst[-1] else: raise StopIteration for i in Foo('Hello'): print (i, end=' ')</pre>	<p>(1) пустая строка (2) H e l l o (3) o l l e H (4) скрипт будет выполняться бесконечно долго (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>
134.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo: def __init__(self, lst): self.lst = lst def __iter__(self): return self def __next__(self): return self.lst.pop() for i in Foo(list(range(4))): print (i+1, end=' ')</pre>	<p>(1) пустая строка (2) 1 2 3 4 (3) 4 3 2 1 и ошибка IndexError (4) скрипт будет выполняться бесконечно долго (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>



135.	Укажите результат выполнения скрипта: class Foo: def __init__(self, lst): self.lst = lst def __iter__(self): return self def __next__(self): if self.lst: return self.lst.pop() else: raise StopIteration for i in Foo(list(range(4))): print(i, end=' ')	(1) 0 1 2 3 (2) 3 2 1 0 (3) скрипт будет выполняться бесконечно долго (4) скрипт выдаст исключение StopIteration (5) скрипт не будет выполнен, т.к. код содержит ошибки
136.	Укажите результат выполнения скрипта: class Foo: def __init__(self, lst): self.lst = lst def __iter__(self): return self def __next__(self): if self.lst: return self.lst.pop(1) else: raise StopIteration for i in Foo(list(range(4))): print(i + 1, end=' ')	(1) 2 3 4 1 (2) 1 2 3 4 (3) 4 3 2 1 (4) 2 3 4 и ошибка IndexError (5) скрипт не будет выполнен, т.к. код содержит ошибки
137.	Укажите результат выполнения скрипта: class Foo: def __init__(self, lst): self.lst = lst self.cur = 0 def __next__(self): if self.lst: self.cur = 1 ^ self.cur return self.lst.pop(self.cur) else: raise StopIteration def __iter__(self): return self for i in Foo(list(range(3))): print(i, end=' ')	(1) 0 1 2 (2) 1 2 0 (3) 0 2 1 (4) 1 0 2 (5) 1 0 и сообщение об ошибке
138.	Укажите результат выполнения скрипта: class Foo: def __init__(self, lst): self.lst = lst self.cur = 1 def __next__(self): if self.lst: self.cur = 1 ^ self.cur return self.lst.pop(self.cur) else: raise StopIteration def __iter__(self): return self for i in Foo(list(range(3))): print(i, end=' ')	(1) 0 1 2 (2) 1 2 0 (3) 0 2 1 (4) 1 0 2 (5) скрипт не будет выполнен, т.к. код содержит ошибки



	<pre>return self for i in Foo(list(range(3))): print (i, end=' ')</pre>	
139.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo: def __init__(self, lst): self.lst = lst self.cur = 1 def __next__(self): if self.lst: self.cur = 1 ^ self.cur return self.lst.pop(-1*self.cur) else: raise StopIteration def __iter__(self): return self for i in Foo(list(range(4))): print (i, end=' ')</pre>	<p>(1) 0 1 2 3 (2) 0 3 1 2 (3) 3 0 2 1 (4) 1 0 3 2 (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>
140.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo: def __init__(self, lst, n): self.n = n self.lst = lst self.cur=0 def __next__(self): if self.lst: self.cur = (self.cur + self.n - 1) % len(self.lst) return self.lst.pop(self.cur) else: raise StopIteration def __iter__(self): return self for i in Foo(list('Hello!'), 3): print (i, end=' ')</pre>	<p>(1) пустая строка (2) H e l l o ! (3) e l ! l H o (4) l ! l e o H (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>
141.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo: def __init__(self, lst): self.lst = lst self.cur = 0 def __next__(self): if self.lst: self.cur = 1 ^ self.cur return self.lst.pop(self.cur) else: raise StopIteration def __iter__(self): return self for i in Foo(list('Hello!')): print (i, end=' ')</pre>	<p>(1) H e l l o ! (2) e H l l ! o (3) H ! e o l l (4) H e o l l l (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>
142.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo: def __init__(self, lst): self.lst = lst</pre>	<p>(1) H e l l o ! (2) H ! e o l l (3) ! H o e l l (4) H e o l l l</p>



	<pre>self.cur = 0 def __next__(self): if self.lst: self.cur = 1 ^ self.cur return self.lst.pop(-1*self.cur) else: raise StopIteration def __iter__(self): return self for i in Foo(list('Hello!')): print (i, end=' ')</pre>	(5) скрипт не будет выполнен, т.к. код содержит ошибки
143.	<p>Скрипт содержит следующий код:</p> <pre>import weakref def meth(): return 1 class Foo(object): val = [meth(), meth(), meth()] try: obj = Foo() s = _____ del obj print (s) except ReferenceError: print ('Error')</pre> <p>Среди приведенных ниже фрагментов укажите все варианты кода, при подстановке которого вместо знаков подчеркивания результатом выполнения скрипта станет строка Error:</p>	<ul style="list-style-type: none">(1) weakref.proxy(obj)(2) weakref.proxy(obj.val)(3) weakref.proxy(obj.val[0])(4) weakref.ref(obj)(5) weakref.ref(obj.val)
144.	<p>Скрипт содержит следующий код:</p> <pre>import weakref class Bar(object): def __repr__(self): return '1' class Foo(object): def __init__(self): self.val = Bar(), Bar(), Bar() try: obj = Foo() s = _____ del obj print (s) except ReferenceError: print ('Error')</pre> <p>Среди приведенных ниже фрагментов укажите все варианты кода, при подстановке которого вместо знаков подчеркивания результатом выполнения скрипта станет строка Error:</p>	<ul style="list-style-type: none">(1) weakref.proxy(obj)(2) weakref.proxy(obj.val)(3) weakref.proxy(obj.val[0])(4) weakref.ref(obj)(5) weakref.ref(obj.val)
145.	<p>Скрипт содержит следующий код:</p> <pre>import weakref class Bar(object): def __repr__(self): return 1 class Foo(object): val = Bar(), Bar(), Bar()</pre>	<ul style="list-style-type: none">(1) weakref.proxy(obj)(2) weakref.proxy(obj.val)(3) weakref.proxy(obj.val[0])(4) weakref.ref(obj)(5) weakref.ref(obj.val)



	<pre>try: obj = Foo() s = _____ del obj print (s) except ReferenceError: print ('Error')</pre> <p>Среди приведенных ниже фрагментов укажите все варианты кода, при подстановке которого вместо знаков подчеркивания результатом выполнения скрипта станет строка Error:</p>	
146.	<pre>Скрипт содержит следующий код: import weakref class Foo(object): def __repr__(self): return 'class' try: s = Foo() s1 = _____ del s print (s1) except ReferenceError: print ('Error')</pre> <p>Среди приведенных ниже фрагментов укажите все варианты кода, при подстановке которого вместо знаков подчеркивания результатом выполнения скрипта станет строка Error:</p>	<ul style="list-style-type: none">(1) s(2) Foo()(3) weakref.proxy(s)(4) weakref.ref(s)
147.	<pre>Скрипт содержит следующий код: import weakref class Foo(object): def __repr__(self): return 'class' try: lst = [Foo(), Foo(), Foo()] s = _____ del lst print (s) except ReferenceError: print ('Error')</pre> <p>Среди приведенных ниже фрагментов укажите все варианты кода, при подстановке которого вместо знаков подчеркивания результатом выполнения скрипта станет строка Error:</p>	<ul style="list-style-type: none">(1) weakref.proxy(lst)(2) weakref.proxy(lst[0])(3) weakref.proxy(Foo())(4) weakref.ref(lst[0])(5) weakref.ref(Foo())
148.	<pre>Скрипт содержит следующий код: import weakref class Foo(object): val = 'Error' try: obj = Foo() s = _____ del obj print (s) except ReferenceError: print ('Error')</pre>	<ul style="list-style-type: none">(1) obj.val(2) weakref.proxy(obj)(3) weakref.proxy(obj.val)(4) weakref.ref(obj)(5) weakref.ref(obj.val)



	Среди приведенных ниже фрагментов укажите все варианты кода, при подстановке которого вместо знаков подчеркивания результатом выполнения скрипта станет строка Erog:	
149.	Чем отличаются в языке Python обычные модули от модулей расширения?	(1) тем, что модули расширения могут состоять из нескольких файлов (2) способом подключения к главному модулю (3) языком программирования, на котором они написаны (4) ничем не отличаются
150.	В языке Python при импортировании пакета, если файл <code>__init__.py</code> отсутствует, то:	(1) будет выдана ошибка, пакет импортирован не будет (2) будет выдано предупреждение и пакет будет импортирован (3) ошибки выдано не будет и пакет импортирован не будет (4) ошибки выдано не будет, пакет будет импортирован, но не будут импортированы его модули
151.	Чем отличаются файлы <code>*.py</code> от файлов <code>*.рус</code> :	(1) ничем (2) языком программирования, на котором они написаны (3) <code>*.рус</code> - это откомпилированные файлы <code>*.py</code> (4) <code>*.рус</code> не могут быть подключены в качестве модулей, в отличие от файлов <code>*.py</code> (5) <code>*.рус</code> не имеют отношения к языку Python
152.	В языке Python модулем называют:	(1) набор файлов, содержащих код на языке Python (2) файл, содержащий определения и другие инструкции на языке Python (3) файл, содержащий определения и другие инструкции на каком-либо языке программирования (4) класс, определенный специальным способом (5) набор классов, связанных наследованием и агрегацией
153.	В языке Python пакетом называют:	(1) набор модулей (2) набор модулей, структурированных определенным образом (3) файл, содержащий определения и другие инструкции на каком-либо языке программирования (4) класс, определенный специальным способом (5) набор классов, связанных наследованием и агрегацией
154.	В языке Python при импортировании модуля с помощью инструкции <code>import M</code> :	(1) в текущее пространство имен добавляется имя модуля и всех его переменных (2) в текущее пространство имен добавляется имена всех его переменных, но имя модуля не добавляется (3) в текущее пространство имен добавляется имена всех его переменных, не начинающихся с подчеркивания. Имя модуля не добавляется (4) в текущее пространство имен добавляется имена имя модуля и всех его переменных, не начинающихся с подчеркивания (5) в текущее пространство имен добавляется только имя модуля
155.	При импортировании модуля если в пространстве	(1) модуль импортируется и заменяет своим именем



	имен уже существует имя импортированного модуля, то:	уже существующее (2) модуль импортируется только в случае, если определенное имя не является именем модуля (3) будет выдана ошибка (4) ошибки выдано не будет, но модуль не импортируется
156.	Встроенный атрибут объекта модуля <code>__dict__</code> :	(1) является атрибутом только для чтения (2) позволяет менять свои элементы по отдельности, но присваивание всего словаря не допустимо (3) поддерживает операцию присваивания, но не позволяет изменять свои элементы по отдельности (4) поддерживает как присваивание всего списка, так и изменение элементов по отдельности (5) поддерживает как удаления элементов списка, также можно удалить сам атрибут
157.	В языке Python при импортировании модуля с помощью инструкции <code>from .. import *</code> :	(1) в текущее пространство имен добавляется имя модуля и всех его переменных (2) в текущее пространство имен добавляется имена всех его переменных, но имя модуля не добавляется (3) в текущее пространство имен добавляется имена всех его переменных, не начинающихся с подчеркивания. Имя модуля не добавляется (4) в текущее пространство имен добавляется имена имя модуля и всех его переменных, не начинающихся с подчеркивания (5) в текущее пространство имен добавляется только имя модуля
158.	В языке Python встроенная функция <code>dir()</code> без аргументов используется для:	(1) получения списка всех имен, доступных в текущей области видимости (2) получения всех имен, доступных в текущем модуле (3) получения имен всех функций, доступных в текущей области видимости (4) получения всех имен модулей, доступных к подключению на текущий момент
159.	Встроенный атрибут объекта модуля <code>__dict__</code> содержит:	(1) таблицу имен всех доступных словарей в модуле (2) таблицу имен всех доступных классов и функций в модуле (3) таблицу всех имен, определенных или переопределенных в модуле (4) таблицу всех имен модуля, в том числе и встроенных
160.	Встроенный атрибут объекта модуля <code>__name__</code> :	(1) содержит имя модуля и является атрибутом только для чтения (2) содержит имя модуля и может быть изменен (3) содержит имя файла, в котором содержится модуль и является атрибутом только для чтения (4) содержит имя файла, в котором содержится модуль и может быть изменен
161.	Файл <code>foo.py</code> содержит следующий код: <pre>def f(x, y): return x+y</pre> Скрипт <code>a.py</code> содержит следующий код: <pre>_1_ print (_2_)</pre> Среди приведенных ниже фрагментов укажите	(1) <code>_1_ : import foo _2_ : f(2, 3)</code> (2) <code>_1_ : import foo as mod _2_ : foo.f(1, 4)</code> (3) <code>_1_ : from foo import * _2_ : foo.f(0, 5)</code> (4) <code>_1_ : from foo import * as fun _2_ : fun(5, 0)</code> (5) <code>_1_ : from foo import f as fun _2_ : fun(3, 2)</code>



	вариант кода, при подстановке которого вместо знаков подчеркивания результатом выполнения скрипта станет строка 5:	
162.	Файл mod.py содержит следующий код: def fun(): return (1, 2, 3) Скрипт a.py содержит следующий код: _1_ print (_2_) Среди приведенных ниже фрагментов укажите вариант кода, при подстановке которого вместо знаков подчеркивания результатом выполнения скрипта станет строка (1, 2, 3):	(1) _1_ : import mod _2_ : mod.fun() (2) _1_ : import mod as fun _2_ : mod.fun() (3) _1_ : from mod import * as fun _2_ : fun() (4) _1_ : from mod import * _2_ : mod.fun() (5) _1_ : from mod import fun as fun _2_ : mod.fun()
163.	Файл module.py содержит следующий код: def function(x, y): return x*y Скрипт a.py содержит следующий код: _1_ print (_2_) Среди приведенных ниже фрагментов укажите все варианты кода, при подстановке которого вместо знаков подчеркивания результатом выполнения скрипта станет строка 12:	(1) _1_ : import module _2_ : function(2, 6) (2) _1_ : import module as module _2_ : module.function(3, 4) (3) _1_ : from module import * as fun _2_ : fun(6, 2) (4) _1_ : from module import function as fun _2_ : module.fun(4, 3) (5) _1_ : from module import function _2_ : function(1, 12)
164.	Файл foo.py содержит следующий код: def f(x): return x**2 Скрипт a.py содержит следующий код: _1_ print (_2_) Среди приведенных ниже фрагментов укажите все варианты кода, при подстановке которого вместо знаков подчеркивания результатом выполнения скрипта станет строка 9	(1) _1_ : import foo _2_ : f(3) (2) _1_ : import foo _2_ : foo.f(x) (3) _1_ : from foo import * _2_ : f(3) (4) _1_ : from foo import * _2_ : foo.f(3) (5) _1_ : from foo import f as fun _2_ : fun(3)
165.	Файл mod.py содержит следующий код: def fun(): return 'Hello' Скрипт a.py содержит следующий код: _1_ print (_2_) Среди приведенных ниже фрагментов укажите все варианты кода, при подстановке которого вместо знаков подчеркивания результатом выполнения скрипта станет строка Hello:	(1) _1_ : import mod _2_ : fun() (2) _1_ : import mod.py _2_ : fun() (3) _1_ : from mod import * _2_ : fun() (4) _1_ : from mod.py import * _2_ : fun() (5) _1_ : from mod.py import * as fun _2_ : fun()
166.	Файл module.py содержит следующий код: def function(x): return [x] Скрипт a.py содержит следующий код: _1_ print (_2_) Среди приведенных ниже фрагментов укажите все варианты кода, при подстановке которого вместо знаков подчеркивания результатом выполнения	(1) _1_ : import module _2_ : module.function(1) (2) _1_ : import module.py _2_ : module.function(1) (3) _1_ : from module import * _2_ : module.function(1) (4) _1_ : from module.py import * _2_ : module.function(1) (5) _1_ : from module import function _2_ : module.function(1)



	скрипта станет строка [1]:	
167.	Файл foo.py содержит следующий код: def f(x): return str(x)+'1' Файл foo2.py содержит следующий код: def f(x): return str(x)+'2' Скрипт a.py содержит следующий код: class Cls (object): def f(self, x): return str(x)+'3' import foo2 as foo import foo foo=Cls() print (foo.f(1)) Укажите результат выполнения скрипта:	(1) 13 (2) 12 (3) 11 (4) скрипт не будет выполнен, т.к. код содержит ошибки
168.	Файл foo.py содержит следующий код: def f(x): return 4*x Файл foo2.py содержит следующий код: def f(x): return 3*x Скрипт a.py содержит следующий код: def f(x): return 2*x from foo import f from foo2 import * as f print (f(1)) Укажите результат выполнения скрипта:	(1) 2 (2) 3 (3) 4 (4) скрипт не будет выполнен, т.к. код содержит ошибки
169.	Файл foo.py содержит следующий код: def f(x): x+=[2] return x Файл foo2.py содержит следующий код: def f(x): x+=[3] return x Скрипт a.py содержит следующий код: import foo as list import foo2 as list print (list.f([1])) Укажите результат выполнения скрипта:	(1) [1] (2) [1, 2] (3) [1, 3] (4) скрипт не будет выполнен, т.к. код содержит ошибки
170.	Файл foo.py содержит следующий код: def f(x): return x**2 Файл foo2.py содержит следующий код: def f(x): return x**3 Скрипт a.py содержит следующий код: def f(x): return x from foo import * from foo2 import *	(1) 3 (2) 9 (3) 27 (4) скрипт не будет выполнен, т.к. код содержит ошибки



	<pre>print (f(3))</pre> Укажите результат выполнения скрипта:	
171.	<p>Файл foo.py содержит следующий код: def f(x): return [x] Файл foo2.py содержит следующий код: def f(x): return (x, Скрипт a.py содержит следующий код: def f(x): return x from foo import * import foo2 print (f(0))</p> Укажите результат выполнения скрипта:	<p>(1) 0 (2) [0] (3) (0, (4) скрипт не будет выполнен, т.к. код содержит ошибки</p>
172.	<p>Файл foo.py содержит следующий код: def f(x): return x**2 Файл foo2.py содержит следующий код: def f(x): return x**3 Скрипт a.py содержит следующий код: class Cls (object): def f(self, x): return x foo=Cls() import foo import foo2 as foo print (foo.f(2))</p> Укажите результат выполнения скрипта:	<p>(1) 2 (2) 4 (3) 8 (4) скрипт не будет выполнен, т.к. код содержит ошибки</p>
173.	<p>Файл ../foo_p/__init__.py содержит следующий код: __all__ = ['foo'] Файл ../foo_p/foo.py содержит следующий код: def f(x): return x**2 Файл ../foo_p/foo1.py содержит следующий код: def f(x): return x**3 Файл ../foo_p/foo2.py содержит следующий код: import foo_p.foo1 def fn(x): return foo1.f(x)*x Скрипт a.py содержит следующий код: _1_ print (_ 2_)</p> Среди приведенных ниже фрагментов укажите все варианты кода, при подстановке которого вместо подчеркивания результатом выполнения скрипта станет строка 8:	<p>(1) _1_ : import foo_p.foo2 _2_ : foo_p.foo2.f(2) (2) _1_ : from foo_p import * _2_ : f(2) (3) _1_ : from foo_p import * _2_ : foo1.f(2) (4) _1_ : from foo_p.foo1 import * _2_ : f(2) (5) _1_ : from foo_p.foo2 import * _2_ : foo1.f(2)</p>
174.	<p>Файл ../foo_p/__init__.py содержит следующий код: __all__ = ['foo1','foo2'] Файл ../foo_p/foo.py содержит следующий код: def f(x): return x**2 Файл ../foo_p/foo1.py содержит следующий код: def f(x): return x**3 Файл ../foo_p/foo2.py содержит следующий код:</p>	<p>(1) _1_ : import foo_p _2_ : foo_p.foo2.fn(2)*foo_p.foo.f(2) (2) _1_ : import foo_p.foo2 as f2 _2_ : f2.fn(2)*f2.f(2) (3) _1_ : from foo_p import * _2_ : foo2.fn(2)*foo.f(2) (4) _1_ : from foo_p import * _2_ : foo2.fn(2)*foo2.f(2) (5) _1_ : from foo_p.foo2 import * _2_ : fn(2)*f(2)</p>



	<pre>from foo_p.foo import f def fn(x): return f(x)*x</pre> <p>Скрипт a.py содержит следующий код:</p> <pre>_1_ print (_2_)</pre> <p>Среди приведенных ниже фрагментов укажите все варианты кода, при подстановке которого вместо подчеркивания результатом выполнения скрипта станет строка 32:</p>	
175.	<p>Файл ../foo_p/__init__.py содержит следующий код:</p> <pre>__all__=['foo1']</pre> <p>Файл ../foo_p/foo.py содержит следующий код:</p> <pre>def f(x): return x**2</pre> <p>Файл ../foo_p/foo1.py содержит следующий код:</p> <pre>def f(x): return x**3</pre> <p>Файл ../foo_p/foo2.py содержит следующий код:</p> <pre>from foo_p import foo1 def fn(x): return foo1.f(x)*x</pre> <p>Скрипт a.py содержит следующий код:</p> <pre>_1_ print (_2_)</pre> <p>Среди приведенных ниже фрагментов укажите все варианты кода, при подстановке которого вместо подчеркивания результатом выполнения скрипта станет строка 8:</p>	<pre>(1) _1_ : import foo_p._2_ : foo_p.foo1.f(2) (2) _1_ : from foo_p import * _2_ : f(2) (3) _1_ : from foo_p import * _2_ : foo1.f(2) (4) _1_ : from foo_p import * _2_ : foo2.f(2) (5) _1_ : from foo_p.foo2 import * _2_ : foo1.f(2)</pre>
176.	<p>Файл ../foo_p/__init__.py содержит следующий код:</p> <pre>__all__=['foo','foo1']</pre> <p>Файл ../foo_p/foo.py содержит следующий код:</p> <pre>def f(x): return x**2</pre> <p>Файл ../foo_p/foo1.py содержит следующий код:</p> <pre>def f(x): return x**3</pre> <p>Файл ../foo_p/foo2.py содержит следующий код:</p> <pre>from foo_p.foo1 import f def fn(x): return f(x)*x</pre> <p>Скрипт a.py содержит следующий код:</p> <pre>_1_ print (_2_)</pre> <p>Среди приведенных ниже фрагментов укажите все варианты кода, при подстановке которого вместо подчеркивания результатом выполнения скрипта станет строка 81:</p>	<pre>(1) _1_ : import foo_p.foo2._2_ : foo_p.foo2.fn(3) (2) _1_ : from foo_p import * _2_ : foo2.fn(3) (3) _1_ : from foo_p import * _2_ : fn(3) (4) _1_ : from foo_p.foo2 import * _2_ : fn(3) (5) _1_ : from foo_p.foo2 import * _2_ : foo2.fn(3)</pre>
177.	<p>Файл ../foo_p/__init__.py содержит следующий код:</p> <pre>__all__=['foo','foo2']</pre> <p>Файл ../foo_p/foo.py содержит следующий код:</p> <pre>def f(x): return x**2</pre> <p>Файл ../foo_p/foo1.py содержит следующий код:</p> <pre>def f(x): return x**3</pre> <p>Файл ../foo_p/foo2.py содержит следующий код:</p> <pre>from foo_p.foo1 import f def fn(x): return f(x)*x</pre> <p>Скрипт a.py содержит следующий код:</p> <pre>_1_ print (_2_)</pre> <p>Среди приведенных ниже фрагментов укажите все варианты кода, при подстановке которого вместо подчеркивания результатом выполнения скрипта станет строка 81:</p>	<pre>(1) _1_ : import foo_p._2_ : foo_p.foo2.fn(2)*foo_p.foo.f(2) (2) _1_ : from foo_p import * _2_ : foo2.fn(2)*foo.f(2) (3) _1_ : from foo_p import * _2_ : fn(2)*f(2) (4) _1_ : from foo_p.foo2 import * _2_ : fn(2)*f(2) (5) _1_ : from foo_p.foo2 import * _2_ : foo2.fn(2)*foo.fn(2)</pre>



	варианты кода, при подстановке которого вместо подчеркивания результатом выполнения скрипта станет строка 64:	
178.	Файл ../foo_p/_init__.py содержит следующий код: __all__=['foo','foo1'] Файл ../foo_p/foo.py содержит следующий код: def f(x): return x**2 Файл ../foo_p/foo1.py содержит следующий код: def f(x): return x**3 Файл ../foo_p/foo2.py содержит следующий код: from foo1 import f def fn(x): return f(x)*x Скрипт a.py содержит следующий код: _1_ print (_2_) Среди приведенных ниже фрагментов укажите все варианты кода, при подстановке которого вместо подчеркивания результатом выполнения скрипта станет строка 32:	(1) _1_ : import foo_p _2_ : foo_p.foo.f(2)*foo_p.foo1.f(2) (2) _1_ : from foo_p import * _2_ : foo.f(2)*foo1.f(2) (3) _1_ : from foo_p import * _2_ : foo_p.foo.f(2)*foo1.f(2) (4) _1_ : from foo_p.foo2 import * _2_ : foo_p.foo.f(2)*foo1.f(2) (5) _1_ : from foo_p.foo2 import * _2_ : foo_p.foo.f(2)*f(2)
179.	При попытке повторной сериализации объектов:	(1) сериализация не производится (2) сериализация производится, но повторная сериализация не изменяет объект (3) сериализация производится, повторная сериализация изменяет объект, но корректное восстановление данных при повторной десериализации не гарантируется (4) сериализация производится, повторная сериализация изменяет объект, корректное восстановление данных при повторной десериализации гарантируется
180.	При сериализации и десериализации объектов произвольного класса необходимо, чтобы:	(1) при десериализации определение класса сериализуемого объекта находилось в том же модуле, где и при сериализации (2) класс сериализуемого объекта был наследником класса object, т.е. сериализацию поддерживают только "новые классы" (3) все переменные объекта должны поддерживать консервацию (4) при сериализации класс объекта должен быть определен в глобальном пространстве имен одного из модулей
181.	При десериализации объекта произвольного класса конструктор класса:	(1) обычно не запускается (2) запускается, только если в классе определен метод __getnewargs__ протокола копирования (3) запускается, только если в классе определены методы __getstate__ и __setstate__ протокола копирования (4) запускается всегда
182.	Сериализация произвольных объектов необходима для:	(1) сохранения произвольных данных на диске (2) передачи произвольных данных по сети (3) увеличения производительности кода (4) улучшения читаемости кода (5) сохранения произвольных данных в базе данных
183.	Модули Pickle и cPickle отличаются	(1) алгоритмом консервирования данных



		(2) интерфейсами, предоставляемых функций (3) производительностью реализуемой сериализации (4) ничем не отличаются (5) Pickle реализует алгоритм на Python, а cPickle на C
184.	Модули Pickle и cPickle сериализуют данные в формат:	(1) последовательности печатных символов utf-8 (2) последовательности печатных символов ASCII (3) последовательность двоичных символов (4) по выбору пользователя: последовательность печатных ASCII или двоичных символов
185.	Файл foo.py содержит следующий код: class Foo(object): def __init__(self, x): self.val=x def __str__(self): return str(self.val) Скрипт a.py содержит следующий код: import pickle, foo o = foo.Foo(list(range(4))) with open('temp.pkl', 'wb') as f: pickle.dump(o, f, pickle.HIGHEST_PROTOCOL) Скрипт b.py содержит следующий код: import pickle with open('temp.pkl', 'rb') as f: x = pickle.load(f) print(x) Укажите результат последовательного выполнения скриптов a.py и b.py:	(1) пустая строка (2) [] (3) [0, 1, 2, 3] (4) скрипт b.py не будет выполнен, т.к. код содержит ошибки
186.	Файл foo.py содержит следующий код: class Foo(object): def __init__(self, x): self.val=x def __str__(self): return str(self.val) Укажите результат выполнения скрипта: import pickle, foo, os o = foo.Foo(list(range(4))) with open('temp.pkl', 'wb') as f: pickle.dump(o, f, pickle.HIGHEST_PROTOCOL) with open('temp.pkl', 'rb') as f: x = pickle.load(f) print(x)	(1) пустая строка (2) [] (3) [0, 1, 2, 3] (4) скрипт не будет выполнен, т.к. код содержит ошибки
187.	Файл foo.py содержит следующий код: class Foo(object): def __init__(self, x): self.val=x def __str__(self): return str(self.val) Скрипт a.py содержит следующий код: import pickle, foo o = foo.Foo(list(range(4))) with open('temp.pkl', 'wb') as f: pickle.dump(o, f, pickle.HIGHEST_PROTOCOL) Скрипт b.py содержит следующий код: import pickle with open('temp.pkl', 'r') as f: x = pickle.load(f) print(x)	(1) пустая строка (2) [] (3) [0, 1, 2, 3] (4) скрипт b.py не будет выполнен, т.к. код содержит ошибки



	Укажите результат последовательного выполнения скриптов a.py и b.py:	
188.	<p>Файл foo.py содержит следующий код:</p> <pre>class Foo(object): def __init__(self, x): self.val=x def __str__(self): return str(self.val)</pre> <p>Скрипт содержит следующий код:</p> <pre>import pickle, foo, os o = foo.Foo(list(range(4))) with open('temp.pkl', 'wb') as f: pickle.dump(o, f, pickle.HIGHEST_PROTOCOL) del foo, o, f with open('temp.pkl', 'rb') as f: x = pickle.load(f) print (str(x))</pre> <p>Укажите результат выполнения скрипта:</p>	<p>(1) пустая строка (2) [] (3) [0, 1, 2, 3] (4) скрипт не будет выполнен, т.к. код содержит ошибки</p>
189.	<p>Файл foo.py содержит следующий код:</p> <pre>class Foo(object): def __init__(self, x): self.val=x def __str__(self): return str(self.val)</pre> <p>Скрипт a.py содержит следующий код:</p> <pre>import pickle, foo, os o = foo.Foo(list(range(4))) with open('temp.pkl', 'wb') as f: pickle.dump(o, f, pickle.HIGHEST_PROTOCOL) f.close()</pre> <p>Скрипт b.py содержит следующий код:</p> <pre>import pickle with open('temp.pkl', 'r') as f: x = pickle.load(f) print (x)</pre> <p>Укажите результат последовательного выполнения скриптов a.py и b.py:</p>	<p>(1) пустая строка (2) [] (3) [0, 1, 2, 3] (4) скрипт b.py не будет выполнен, т.к. код содержит ошибки</p>
190.	<p>Файл foo.py содержит следующий код:</p> <pre>class Foo(object): def __init__(self, x): self.val=x def __str__(self): return str(self.val)</pre> <p>Скрипт a.py содержит следующий код:</p> <pre>import pickle, foo, os o = foo.Foo(list(range(4))) with open('temp.pkl', 'wb') as f: pickle.dump(o, f, pickle.HIGHEST_PROTOCOL) f.close()</pre> <p>Скрипт b.py содержит следующий код:</p> <pre>import pickle with open('temp.pkl', 'rb') as f: x = pickle.load(f) print (x)</pre> <p>Укажите результат выполнения скрипта b.py:</p>	<p>(1) пустая строка (2) [] (3) [0, 1, 2, 3] (4) скрипт b.py не будет выполнен, т.к. код содержит ошибки</p>
191.	<p>Укажите результат выполнения скрипта:</p> <pre>import pickle</pre>	<p>(1) [] False False (2) [1, 2, [...]] False False</p>



	<pre>a = [1, 2] a.append([1, 2]) with open('temp.pkl', 'wb') as f: pickle.dump(a, f, pickle.HIGHEST_PROTOCOL) with open('temp.pkl', 'rb') as f: b, c = pickle.load(f) print(c) print(b is c, b == c)</pre>	<p>(3) [1, 2, [...]] False True (4) [1, 2, [...]] True True (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>
192.	<p>Укажите результат выполнения скрипта:</p> <pre>import pickle a = [1, 2] b = [3, 4] a.append(b) b.append(a) with open('temp.pkl', 'w') as f: pickle.dump(a, f, pickle.HIGHEST_PROTOCOL) with open('temp.pkl', 'r') as f: c = pickle.load(f) with open('temp.pkl', 'w') as f: pickle.dump(b, f, pickle.HIGHEST_PROTOCOL) with open('temp.pkl', 'r') as f: d = pickle.load(f) print(c, d)</pre>	<p>(1) False False (2) False True (3) True False (4) True True (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>
193.	<p>Укажите результат выполнения скрипта:</p> <pre>import pickle a = [1, 2] a.append(a) with open('temp.pkl', 'wb') as f: pickle.dump(a, f, pickle.HIGHEST_PROTOCOL) with open('temp.pkl', 'r') as f: b = pickle.load(f) with open('temp.pkl', 'wb') as f: pickle.dump(a[2], f, pickle.HIGHEST_PROTOCOL) with open('temp.pkl', 'r') as f: c = pickle.load(f) print(b, c)</pre>	<p>(1) [] False False (2) [1, 2, [...]] False False (3) [1, 2, [...]] True False (4) [1, 2, [...]] True True (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>
194.	<p>Укажите результат выполнения скрипта:</p> <pre>import pickle a = [1, 2] b = [3, 4] a.append(b) b.append(a) with open('temp.pkl', 'wb') as f: pickle.dump(a, f, pickle.HIGHEST_PROTOCOL) with open('temp.pkl', 'rb') as f: c = pickle.load(f) print(c) print(c is a, c[2] is b)</pre>	<p>(1) [] False False (2) [1, 2, [3, 4, [...]]] False False (3) [1, 2, [3, 4, [...]]] True False (4) [1, 2, [3, 4, [...]]] True True (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>
195.	<p>Укажите результат выполнения скрипта:</p> <pre>import pickle a = [1, 2] a.append(a)</pre>	<p>(1) [] False False (2) [1, 2, [...]] False False (3) [1, 2, [...]] False True (4) [1, 2, [...]] True True</p>



	<pre>with open('temp.pkl', 'wb') as f: pickle.dump(a, f, pickle.HIGHEST_PROTOCOL) with open('temp.pkl', 'rb') as f: b= pickle.load(f) print (b) print (b is a, b[2] is b)</pre>	(5) скрипт не будет выполнен, т.к. код содержит ошибки
196.	<p>Укажите результат выполнения скрипта:</p> <pre>import pickle a = [1, 2] b = [3, 4] a.append(b) b.append(a) with open('temp.pkl', 'wb') as f: pickle.dump((a,b), f, pickle.HIGHEST_PROTOCOL) with open('temp.pkl', 'rb') as f: c,d= pickle.load(f) print (c) print (c[2] is d, c[2] == d)</pre>	(1) [] False False (2) [1, 2, [3, 4, [...]]] False False (3) [1, 2, [3, 4, [...]]] True False (4) [1, 2, [3, 4, [...]]] True True (5) скрипт не будет выполнен, т.к. код содержит ошибки
197.	<p>Укажите результат выполнения скрипта:</p> <pre>class str(object): def __new__(self, lst): res=0 for i in lst: res+=1 return res print (str(range(5, 8)))</pre>	(1) пустая строка (2) [5, 6, 7] (3) 3 (4) скрипт не будет выполнен, т.к. код содержит ошибки
198.	<p>Укажите результат выполнения скрипта:</p> <pre>def add (x, y): return x+y class add(object): def __call__(self, x, y): return (x+y)*2 f = add() print (f(1, 2))</pre>	(1) пустая строка (2) 3 (3) 6 (4) скрипт не будет выполнен, т.к. код содержит ошибки
199.	<p>Укажите результат выполнения скрипта:</p> <pre>def add (x, y): return x+y class add(object): def __init__(self, x, y): return (x+y)*2 print (add(1, 2))</pre>	(1) пустая строка (2) 3 (3) 6 (4) скрипт не будет выполнен, т.к. код содержит ошибки
200.	<p>Укажите результат выполнения скрипта:</p> <pre>class len(object): def __call__(self, lst): res=0 for i in lst: res+=1 return res print (len(range(5, 8)))</pre>	(1) пустая строка (2) 0 (3) 3 (4) скрипт не будет выполнен, т.к. код содержит ошибки
201.	<p>Укажите результат выполнения скрипта:</p> <pre>class len(object):</pre>	(1) пустая строка (2) 0



	<pre>def __call__(self, lst): res=0 for i in lst: res+=1 return res foo = len() print (foo(range(8)))</pre>	<p>(3) 8 (4) скрипт не будет выполнен, т.к. код содержит ошибки</p>
202.	<p>Укажите результат выполнения скрипта:</p> <pre>class str(object): def __init__(self, lst): res=0 for i in lst: res+=1 return res print (str(range(0, 4)))</pre>	<p>(1) пустая строка (2) [0, 1, 2, 3] (3) 4 (4) скрипт не будет выполнен, т.к. код содержит ошибки</p>
203.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo(object): def __init__(self, v): self.__val__ = v def __add__(self, y): return Foo(self.__val__ + y - 1) def __repr__(self): return str(self.__val__) x = Foo(1) print (x+2, 2+x)</pre>	<p>(1) 2 2 (2) 2 3 (3) 3 2 (4) 3 3 (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>
204.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo(object): def __init__(self, v): self.__val__ = v def __isub__(self, y): return self.__val__ - y + 1 def __sub__(self, y): return self.__val__ - y + 1 def __repr__(self): return str(self.__val__) x = Foo(3) x-=2 print (x, x-1)</pre>	<p>(1) 1 0 (2) 1 1 (3) 2 1 (4) 2 2 (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>
205.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo(object): def __init__(self, v): self.__val__ = v def __rsub__(self, y): return self.__val__ - y + 1 def __isub__(self, y): return Foo(self.__val__ - y + 1) def __repr__(self): return str(self.__val__) x = Foo(5) x-=3 print (x, x-1)</pre>	<p>(1) 2 1 (2) 3 2 (3) 3 3 (4) 2 2 (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>
206.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo(int): def __init__(self, v):</pre>	<p>(1) <class '__main__.Foo'> <class '__main__.Foo'> (2) <class '__main__.Foo'> <class 'int'> (3) <class 'int'> <class '__main__.Foo'></p>



	<pre>self.__val__ = v def __mul__(self, y): self.__val__ *= y return self x = Foo(2) print (type(x*5), type(5*x))</pre>	<p>(4) <class 'int'> <class 'int'> (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>
207.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo(int): def __init__(self, v): self.__val__ = v def __imul__(self, y): self.__val__ *= y return self def __rmul__(self, y): self.__val__ *= y return self def __repr__(self): return str(self.__val__) x = Foo(2) print (x*3, 3*x)</pre>	<p>(1) 2 2 (2) 2 6 (3) 6 2 (4) 6 6 (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>
208.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo(int): def __init__(self, v): self.__val__ = v def __add__(self, y): return Foo(self.__val__ + y - 1) def __repr__(self): return str(self.__val__) x = Foo(1) print (x+4, 4+x)</pre>	<p>(1) 4 4 (2) 4 5 (3) 5 4 (4) 5 5 (5) скрипт не будет выполнен, т.к. код содержит ошибки</p>
209.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo(object): @classmethod def method1(): return 1 @staticmethod def method2(): return 2 def method3(): return 3 method3 = staticmethod(method3) print (Foo.method1(), Foo.method2(), Foo.method3())</pre>	<p>(1) скрипт выведет сообщение об ошибке (2) скрипт выведет 1 и сообщение об ошибке (3) скрипт выведет 1 2 и сообщение об ошибке (4) 1 2 3</p>
210.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo(object): @classmethod def method1(cls): return 1 @staticmethod def method2(cls): return 2 def method3(cls): return 3 method3 = classmethod(method3)</pre>	<p>(1) скрипт выведет сообщение об ошибке (2) скрипт выведет 1 и сообщение об ошибке (3) скрипт выведет 1 2 и сообщение об ошибке (4) 1 2 3</p>



	<pre>print (Foo.method1(), end=' ') print (Foo.method2(), end=' ') print (Foo.method3(), end=' ')</pre>	
211.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo(object): @classmethod def method1(cls): return 1 @staticmethod def method2(): return 2 def method3(cls): return 3 method3 = classmethod(method3) print (Foo.method1(), end=' ') print (Foo.method2(), end=' ') print (Foo.method3(), end=' ')</pre>	<p>(1) скрипт выведет сообщение об ошибке (2) скрипт выведет 1 и сообщение об ошибке (3) скрипт выведет 1 2 и сообщение об ошибке (4) 1 2 3</p>
212.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo(object): @staticmethod def method1(): return 1 def method2(): return 2 def method3(): return 3 method2 = staticmethod(method2) print (Foo.method1(), Foo.method2(), Foo.method3())</pre>	<p>(1) скрипт выведет сообщение об ошибке (2) скрипт выведет 1 и сообщение об ошибке (3) скрипт выведет 1 2 и сообщение об ошибке (4) 1 2 3</p>
213.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo(object): method2 = staticmethod(method2) @staticmethod def method1(): return 1 def method2(): return 2 def method3(): return 3 print (Foo.method1(), Foo.method2(), Foo.method3())</pre>	<p>(1) скрипт выведет сообщение об ошибке (2) скрипт выведет 1 и сообщение об ошибке (3) скрипт выведет 1 2 и сообщение об ошибке (4) 1 2 3</p>
214.	<p>Укажите результат выполнения скрипта:</p> <pre>class Foo(object): @staticmethod def method1(): return 1 def method2(): return 2 def method3(): return 3 method2 = classmethod(method2) method3 = staticmethod(method3) print (Foo.method1()) print (Foo.method2()) print (Foo.method3())</pre>	<p>(1) скрипт выведет сообщение об ошибке (2) скрипт выведет 1 и сообщение об ошибке (3) скрипт выведет 1 2 и сообщение об ошибке (4) 1 2 3</p>



215.	Укажите результат выполнения скрипта: <pre>class Foo(int): @classmethod def method(cls, val=3): if cls is Foo: return val+2 else: return val+3 @staticmethod def method1(cls, val=3): if cls is Foo: return val+2 else: return val+3 print (Foo.method(1), Foo.method1(1))</pre>	(1) 4 3 (2) 3 4 (3) 4 4 (4) 3 6 (5) 6 3
216.	Укажите результат выполнения скрипта: <pre>class Foo(int): @classmethod def method(cls, val=3): if isinstance (cls, Foo): return val+2 else: return val+3 @staticmethod def method1(cls, val=3): if isinstance (cls, Foo): return val+2 else: return val+3 f=Foo() print (f.method(1), f.method1(1))</pre>	(1) 3 3 (2) 6 4 (3) 4 6 (4) 3 6 (5) 6 3
217.	Укажите результат выполнения скрипта: <pre>class Foo(int): def method(cls, val=3): if cls is Foo: return val+2 else: return val+3 method=classmethod(method) method=staticmethod(method) print (Foo.method(1), Foo().method1(1))</pre>	(1) 4 3 (2) 3 4 (3) 3 6 (4) 6 3 (5) скрипт не будет выполнен, т.к. код содержит ошибки
218.	Укажите результат выполнения скрипта: <pre>class Foo(object): def __init__(self): self.__val__=2 def method(cls, val): if isinstance (cls, Foo): return val+cls.__val__ else: return val+1 method = classmethod(method) f=Foo() print (Foo.method(1), f.method(1))</pre>	(1) 1 1 (2) 2 1 (3) 2 2 (4) скрипт не будет выполнен, т.к. код содержит ошибки



219.	Укажите результат выполнения скрипта: <pre>class Foo(int): @staticmethod def method(cls, val=3): if cls == Foo: return val+2 else: return val+1 f=Foo() print (Foo.method(1), f.method(1))</pre>	(1) 2 2 (2) 2 3 (3) 3 4 (4) 4 4 (5) скрипт не будет выполнен, т.к. код содержит ошибки
220.	Укажите результат выполнения скрипта: <pre>class Foo(int): @staticmethod def method(cls, val=3): if cls == Foo: return val+2 else: return val+1 method = classmethod(method) f=Foo() print (Foo.method(1), f.method(1))</pre>	(1) 2 2 (2) 2 3 (3) 3 4 (4) 4 4 (5) скрипт не будет выполнен, т.к. код содержит ошибки
221.	Методы класса содержатся в пространстве имен:	(1) класса (2) класса и экземпляров класса (3) класса, его метакласса и экземплярах класса (4) класса и метакласса (5) в глобальном пространстве имен
222.	Метакласс класса C можно определить по:	(1) атрибуту класса C. __metaclass__ (2) атрибуту класса C. __class__ (3) вызвав функцию type(C) (4) вызвав функцию type(C()) (5) вызвав функцию class(C)
223.	Метакласс M для класса C можно задать следующим образом:	(1) при определении класса установить атрибут класса C __metaclass__ равным M (2) при определении класса установить атрибут класса C __class__ равным M (3) при определении класса установить атрибут класса C __type__ равным M (4) наследовать класс C от metaclass=M
224.	Метакласс - это:	(1) абстрактный класс (2) класс, объекты которого являются объектами других классов (3) класс, объекты которого являются классами (4) класс, объекты которого являются функциями (5) верхний класс в иерархии классов
225.	Метаклассами являются следующие встроенные классы:	(1) object (2) type (3) metaclass (4) в языке Python нет встроенных метаклассов
226.	Методы, определенные в метаклассе содержатся в пространстве имен:	(1) метакласса (2) метакласса и класса (3) метакласса, класса и экземпляра класса (4) класса и экземпляра класса (5) в метаклассе нельзя определять методы



227.	<p>Скрипт содержит следующий код:</p> <pre>def foo(f): class X(object): pass return X def method(x, y): return x+y Cs = foo(method) o = Cs() print (o.method(2, 3))</pre> <p>Среди приведенных ниже фрагментов укажите все варианты кода, при подстановке которого вместо знаков подчеркивания результатом выполнения скрипта станет строка 5:</p>	<ol style="list-style-type: none">(1) Cs.__dict__['method']=method(2) Cs.method=method(3) Cs.method=lambda i, x, y: x+y(4) o.__dict__['method']=method(5) o.method=method
228.	<p>Скрипт содержит следующий код:</p> <pre>def foo(f): class X(object): pass return X def method(lst): return len(lst) Cs = foo(method) print (Cs.method(range(7)))</pre> <p>Среди приведенных ниже фрагментов укажите все варианты кода, при подстановке которого вместо знаков подчеркивания результатом выполнения скрипта станет строка 7:</p>	<ol style="list-style-type: none">(1) setattr(X, 'method', f)(2) setattr(X, 'method', classmethod(lambda i: len(i)))(3) X.__dict__['method']=classmethod(f)(4) X.method=lambda i: len(i)(5) X.method=classmethod(f)
229.	<p>Скрипт содержит следующий код:</p> <pre>def foo(f): class X(object): pass return X def method(lst): return len(lst) Cs = foo(method) print (Cs.method(range(3)))</pre> <p>Среди приведенных ниже фрагментов укажите все варианты кода, при подстановке которого вместо знаков подчеркивания результатом выполнения скрипта станет строка 3:</p>	<ol style="list-style-type: none">(1) setattr(Cs, 'method', classmethod(method))(2) setattr(Cs, 'method', classmethod(lambda i, x: method(x)))(3) Cs.__dict__['method']=lambda i, x: method(x)(4) Cs.__dict__['method']=classmethod(method)(5) Cs.method=method
230.	<p>Скрипт содержит следующий код:</p> <pre>def foo(f): class X(object): pass return X def method(self): return 'hello' Cs = foo(method) o = Cs() print (o.method(), type(o))</pre>	<ol style="list-style-type: none">(1) setattr(X, f.__name__, f)(2) setattr(X, method.__name__, f)(3) setattr(X, 'f', f)(4) setattr(X, 'method', method)(5) setattr(X, 'method', f)



	Среди приведенных ниже фрагментов укажите все варианты кода, при подстановке которого вместо знаков подчеркивания результатом выполнения скрипта станет строка <code>hello<class ' __main__ .X'></code> :	
231.	<p>Скрипт содержит следующий код:</p> <pre>def foo(f): class X(object): pass return X def method(): return 'hello' Cs = foo(method) o = Cs() print (o.method())</pre> <p>Среди приведенных ниже фрагментов укажите все варианты кода, при подстановке которого вместо знаков подчеркивания результатом выполнения скрипта станет строка <code>hello</code>:</p>	<ol style="list-style-type: none">(1) <code>setattr(Cs, 'method', method)</code>(2) <code>setattr(Cs, 'method', lambda i: 'hello')</code>(3) <code>setattr(o, 'method', method)</code>(4) <code>setattr(o, 'method', lambda i: 'hello')</code>(5) <code>setattr(Cs, method.__name__, method)</code>
232.	<p>Скрипт содержит следующий код:</p> <pre>def foo(f): class X(object): pass return X def method(x, y): return x+y Cs = foo(method) o = Cs() print (o.method(1, 2))</pre> <p>Среди приведенных ниже фрагментов укажите все варианты кода, при подстановке которого вместо знаков подчеркивания результатом выполнения скрипта станет строка <code>3</code>:</p>	<ol style="list-style-type: none">(1) <code>setattr(X, f.__name__, lambda i, x, y : f(x,y))</code>(2) <code>setattr(X, method.__name__, f)</code>(3) <code>X.__dict__["method"] = f</code>(4) <code>X.__dict__[f.__name__] = lambda i, x, y : f(x,y)</code>(5) <code>X.method = f</code>
233.	<p>Укажите результат выполнения скрипта:</p> <pre>class foo(type): val=1 class A(object): val=0 __class__ =foo print (type(A), A.val)</pre>	<ol style="list-style-type: none">(1) <code><class ' __main__ .foo'> 0</code>(2) <code><class ' __main__ .foo'> 1</code>(3) <code><class 'type'> 0</code>(4) <code><class 'type'> 1</code>(5) скрипт не будет выполнен, т.к. код содержит ошибки
234.	<p>Укажите результат выполнения скрипта:</p> <pre>class foo(type): val=1 class bar(type): val=2 class A(object, metaclass=foo): val=-1 class B(object, metaclass=bar): val=-2 class C(A,B): pass print (type(C), C.val)</pre>	<ol style="list-style-type: none">(1) <code><class ' __main__ .foo'> -1</code>(2) <code><class ' __main__ .foo'> 1</code>(3) <code><class ' __main__ .bar'> -2</code>(4) <code><class ' __main__ .bar'> 2</code>(5) скрипт не будет выполнен, т.к. код содержит ошибки



235.	Укажите результат выполнения скрипта: class foo(type): pass class bar(type): pass class A(object, metaclass=foo): val=-1 class B(object, metaclass=bar): val=-2 class baz (foo, bar): pass class C(A, B, metaclass=bar): pass print (type(C), C.val)	(1) <class '__main__.baz'> -1 (2) <class '__main__.foo'> -1 (3) <class '__main__.foo'> -1 (4) <class '__main__.bar'> -2 (5) скрипт не будет выполнен, т.к. код содержит ошибки
236.	Укажите результат выполнения скрипта: class foo(type): def __new__(cls, name, bases, dict): return type.__new__(cls, name, bases, dict) o = foo('X', (), {}) print (type(o), o)	(1) <class '__main__.foo'> <class '__main__.foo'> (2) <class '__main__.foo'> <class '__main__.X'> (3) <class '__main__.X'> <class '__main__.foo'> (4) <class '__main__.X'> <class '__main__.X'> (5) скрипт не будет выполнен, т.к. код содержит ошибки
237.	Укажите результат выполнения скрипта: class foo(type): def __new__(cls, name, bases, dict, defval=0): obj = type.__new__(cls, name, bases, dict) obj.val=defval return obj Cls = foo('X', (), {}) class SCls (Cls): pass print (SCls.val, type(SCls))	(1) 0 <type 'type'> (2) 2 <type 'type'> (3) 0 <class '__main__.foo'> (4) 2 <class '__main__.foo'> (5) скрипт не будет выполнен, т.к. код содержит ошибки
238.	Укажите результат выполнения скрипта: class foo(type): val=1 class A(object): val=0 __metaclass__=foo print (type(A), A.val)	(1) <class '__main__.foo'> 0 (2) <class '__main__.foo'> 1 (3) <class 'type'> 0 (4) <class 'type'> 1 (5) скрипт не будет выполнен, т.к. код содержит ошибки

Примерные вопросы на экзамен:

1. Основные особенности языка python
2. Типы данных
3. Байткод python
4. Базовые библиотеки языка
5. Структуры данных, особенности
6. Функциональное программирование
7. Декораторы, генераторы
8. Асинхронное программирование



9. Многопоточное программирование
10. Global Interpreter Lock
11. Особенности ООП в python
12. Контекстный менеджер
13. Дескрипторы, метаклассы
14. Стандарты кодирования python
15. Методы рефакторинга
16. ORM в python
17. Паттерн repository
18. Веб фреймворк flask, мотивация использования
19. Веб фреймворк bottle, мотивация использования
20. Веб фреймворк django, мотивация использования
21. Маршрутизация в Django
22. Шаблонизаторы



4. Порядок проведения и критерии оценивания промежуточной аттестации

4.1. Порядок проведения промежуточной аттестации

Экзамен проводится в виде устного ответа на одну из выбранных тем. Студент должен дать развернутый ответ на поставленный вопрос, раскрыть суть поставленной темы, привести практические примеры. Продолжительность ответа – 15 минут.

4.2. Критерии оценивания промежуточной аттестации по видам оценочных средств

4.2.1. Критерии оценивания опроса

Обучающийся должен глубоко и полно владеть содержанием учебного материала; уметь связывать теорию с практикой, теоретические выводы подтверждать примерами, фактами, данными научных исследований. Обучающийся знает и воспроизводит основные определения, связанные с программированием на python, понимает принципы их (устройств) функционирования; демонстрирует навыки написания веб сервисов, для которых нет очевидного решения; оценивает достоверность полученного результата, может обосновать принимаемые при проектировании решения и предлагать альтернативные пути решения и оценить их преимущества и недостатки.

Обучающийся может получить «не удовлетворительно», если он не может воспроизвести основные определения, не знает назначение и общие принципы программирования на python; не способен применять изученные методы для выполнения типового задания.

4.3. Результаты промежуточной аттестации и уровни сформированности компетенций

Для подведения итогов студент должен сдать 5 практических работ и ответить на итоговый вопрос.

Отлично: Величина рейтинга обучающегося по дисциплине 85...100 %.

Хорошо: Величина рейтинга обучающегося по дисциплине 75...84 %.

Удовлетворительно: Величина рейтинга обучающегося по дисциплине 60...74. %

Неудовлетворительно: Величина рейтинга обучающегося по дисциплине 0...59 %.

Особенности проведения процедуры оценивания результатов обучения инвалидов и лиц с ограниченными возможностями здоровья обозначены в рабочей программе дисциплины (модуля).

Уровни сформированности компетенций определяется следующим образом:

1. Высокий уровень сформированности компетенций соответствует оценке отлично:
 - предполагает формирование компетенций на высоком уровне;
 - знание теоретических разделов изучаемой дисциплины на уровне не ниже оценки отлично;
 - студент умеет применять на практике знания, полученные в рамках изучения дисциплины
 - формируются навыки использования теоретических и практических разделов дисциплины для решения задач профессиональной деятельности;



2. Средний уровень соответствует оценке хорошо:
 - предполагает формирование компетенций на среднем уровне;
 - знание теоретических разделов изучаемой дисциплины на уровне не ниже оценки хорошо;
 - студент умеет применять знания, полученные в рамках изучения дисциплины, для решения задач профессиональной деятельности;
3. Базовый уровень соответствует оценке удовлетворительно:
 - предполагает формирование компетенций на базовом уровне;
 - знание теоретических разделов изучаемой дисциплины на уровне не ниже оценки удовлетворительно;
4. Недостаточный уровень соответствует оценке неудовлетворительно.