SKILLFACTORY

Тонкости обработки исключений.

Собственные классы исключений.

Исключения – это такие особенные классы, которые как и любые классы можно наследовать. Если вы хотите ловить несколько исключений – то сначала ловите потомков, а потом родителей, чтобы ничего не упустить.
Чтобы создать собственный класс, нужно просто написать пустой класс и наследовать его от класса Exception, этого будет достаточно.
Не обязательно отлавливать сам класс, при необходимости можно отлавливать его родителя, это тоже будет работать, но вы можете упустить важную информацию.

Синтаксис собственного класса исключения:
 	
`class MyException(Exception):`
`  pass`
 	
Вызывается с помощью raise и ведёт себя также, как и другие исключения

Fullstack разработчик на Python

Работа с импортом

Обычный импорт	import *имя модуля*
Импорт модуля из каталога	import *путь к каталогу*.*имя модуля*
Импорт всех структур из модуля	from *имя модуля* import *
Обращение к элементам модуля	*имя модуля*.*структура для обращения* Если вы работаете в ide, то она сама подсветит вам возможные варианты для вызова.
Импорт модуля под другим именем	import *имя модуля* as *имя для обращения*
Основные принципы описания собственного модуля	Вся логиика должна быть вынесена в функции или классы. Если вы хотите выполнять какой-то код в самом модуле, то лучше всего вынести его в функцию main() и запускать только при условии что модуль является запускаемым файлом.
if __name__ == '__main__':	Проверяет что модуль является запускаемым файлом.

Модуль
Модули и импорт. Работа с файлами и данными.

Работа с файлами

Функция для открытия файла	f = open('*имя файла*', '*режим для записи или чтения, текстовый или бинарный*')
Режимы открытия файла	w – переписать файл или создать новый a – записать в конец файла r – прочитать информацию из файла к ним можно так же добавить: t – режим чтения текстовых файлов b – режим чтения двоичных файлов
Принципы работы с файлами	При окончании работы с файлами, чтобы он не блокировался и не терял записанную информацию его обязательно надо закрывать методом .close()

Менеджер контекста with	Удобно применять при работе с файлами например:
	
	`with open('f.txt', 'w') as f:`
	`  *блок кода для работы с файлом f*`

Контекстные менеджеры

Контекстный менеджер вызывается с помощью ключевого слова with. Он может возвращать или не возвращать объект для работы. Например если контекстный менеджер подразумевает работу с каким-либо объектом, то надо добавить в запись: “as *var*” где var - имя переменной в данном контексте (извините за коламбур).

Можно легко писать собственные контекстные менеджеры на основе классов. Для этого мы пишем класс со специальными методами

```
def __enter__(self, …)
```

и

```
def __exit__(self, exc_type, exc_val, exc_tb)
```

__enter__ - вызывается при входе в контекстный менеджер
__exit__ - вызывается при выходе из контекстного менеджера

Пример контекстных менеджеров на основе генераторов:

```
from contextlib import contextmanager

@contextmanager # оборачиваем функцию в декоратор contextmanager
def timer():
    *код при входе*
    yield # если вам нужно что-то вернуть через контекстный менеджер, просто вставьте этот объект сюда.
```