**Программирование, предметная область, обучение.**

**Кредо**

При изучении программирования приходится решать три типичные задачи:

1. Какие языки изучать
2. Какие примеры предлагать при изучении
3. Понимание программирования на таком уровне, что бы переход с одного языка на другой не вызывал затруднений.

Со всеми тремя задачами, ныне, определенно, есть проблемы. Типичное их решение – а на чем пишут соседи, друзья, знакомые, а что предлагает (чаще навязывает ) Microsoft, а какие существуют вакансии (не догадываясь, что список вакансий, часто виртуальный, формируют, как рекламный проект продвижения продукции, часто очень спорного качества), примеры же берутся из сети, с перегретым интересом к компьютерным играм и сайтостроительства, третья же задача вообще рассматривают как подрыв авторитета нынешних преподавателей вузов IT направления. Многие из них не различают программирование, как навык, и знание языка программирования как частный случай этого навыка. В чистом виде это различие реализуется в свободе перехода с языка на язык. Для того, что бы говорить на разных языках необходимо не быть немым, как минимум. Понимание базовых моделей вычислений, это и есть отсутствие немоты в программировании.

Вместе с тем, решение этих задач на приемлемом уровне существует и уже давно, естественно, корректируется с учетом текущих реалий. А текущие реалии таковы, что изучение и понимание дисциплин естественно – научного плана (физика, математика) и соответствующие навыки находится на катастрофически низком уровне. Эти навыки, типичные для школы прошлого, замещены ныне исключительно визуальным восприятием, без активного участия размышления. Поскольку язык математики универсален в достаточно высокой степени, то изучение программирования, как это было в прошлом, на примерах из математики, как универсальной предметной области, уже **не работает**. Любой пример для программирования предполагает знание предметной области, более того, эта область должна быть известной и понятной, не вызывать внутреннего напряжения, иначе – отказ от изучения или переход на метод “кое как и поскорее отделаться”, этот метод вклинивается в психологию как вирус, и надолго, такой программист скорее опасен, чем полезен.

Вместе с тем, игнорировать реалии глупо и бесперспективно. Ныне изучение, и программирования в том числе, должно включать визуальный ряд восприятия при обучении. Эта установка должна сработать как при мотивации, так и при проверке результатов работы, благо, что аномалии визуального поведения позволяют обнаружить ошибку кода без утомительного анализа исходного текста.

Но это мы разобрали всего лишь со вторым вопросом. Но как быть с 1-ым и 3-им? Полный ответ потребовал бы уже не статьи, а минимум сборника или даже отдельной монографии. Но нам пока нужно схватить суть дела. Предположим у вас есть равенство:

**A = B**

Проще простого, но что здесь утверждается? ***Пользуемся только здравым смыслом и вниманием даже к тому, что считается совершенно банальным***. Ох, как это нужно нам сейчас и в будущем!

Итак, этот очень простой текст утверждает:

1. некоторые вещи (сущности), обозначенные буквами А и B совпадают (или равны друг другу) и всегда, то есть мы знаем смысл значка “=”. Но, внимание, если они совпадают, то почему обозначены разными буквами? Равны и совпадают, это одно и то же или нет? Равны, это когда есть некоторая мера измерения, и значения меры совпадают (наложением) а совпадение проверяется уже без меры (есть ли что- либо кроме совпадения?)
2. Если одинаковые сущности обозначены разными буквами, то тогда это равенство не сущностей, а совпадение области значений имен. Есть имя Василий, но значением этого имени (денотат) будет какой то конкретный человек. И буквы A и B имеют смысл имени, но никак не вещи или сущности (мысль, допустим, не является вещью, но является сущностью). Как в жизни, у вас есть имя и фамилия по паспорту (это A), а есть сокращенный вариант для друзей (это B) . И эти два вида имен обозначают одного и того же человека (A=B).
3. сущности А и B как то связаны между собой и эта связь выписана значком “=”, то есть, в данном случае, уже более вероятно понимание этого равенства, как определение знака “=”. Типичный пример из обычной жизни, например, принуждение к миру, увы, актуальное сегодня. В зависимости от ситуации при принуждении (война на уничтожение) смысл у знака может и не быть, то есть пустое множество.

Обратите внимание, как много появилось вариантов толкования даже такого простого выражения. А как быть с более сложными выражениями? Выход только один – ввести перечень безусловных правил понимания текста (синтаксис и семантика) и желательно обосновать, точно и однозначно выбор списка этих правил (этап разработки языка), а это обоснование уже вычислительная модель, лежащая в основе языка, причем любого (модели, конечно же, разные!).

Как это проявляется конкретно в языке программирования? Если это равенство рассматривать в языках Python и С++, то смысл их будет дополнителен к друг другу (если этих смыслов было только 2, то мы бы сказали не дополнительные, а противоположные), а именно:

1. В равенстве A = B вся информация о сущности находится на стороне B, а буковка A это либо синоним B, либо указывает место нахождения B. Это вариант Python, приведение типов не нужно, типы вычисляются по правой части выражения. В тексте программы типы явно нигде не указываются, поскольку определяются правой частью.
2. В равенстве A = B, вся информация о сущности находится на стороне A, и нужен дополнительный анализ B, что и реализуется приведениями типов и является существенной частью собеседований на знание С++ при приеме на работу. Существенным подспорьем для такого анализа является обязательным указанием типа переменной (в С++ 11 уже несколько иначе)

Вот мы и готовы дать ответ на вопрос о языках. Языков должно быть минимум 2 и они должны быть дополнительны друг другу. Перечислять возможные пары мы не будем, достаточно одной пары из этих 2-х языков С++ и Python.

Попутно, мы уже догадываемся, что обнаружение и знакомство на практике с вычислительной моделью, лежащей в основе языка, в общих пока чертах, позволит нам в дальнейшем свободно прогуливаться, при желании, физических ограничений немоты нет, в пространстве языков программирования. Важным здесь является то, что ***базовые языки изучения должны быть дополнительны друг к другу***. В этом нет ничего нового или необычного. В обычной жизни, как и на суде, есть адвокаты и прокуроры, которые позволяют рассмотреть предмет разбирательства с противоположных сторон. Здесь все аналогично.

**Что мы выбираем?**

Всем перечисленным выше озабоченностям и замечаниям в наилучшей степени отвечает сочетание Cinema 4D, как практически неограниченный набор предметных областей и языковый интерфейс к этому пакету на C++ и Python.

При этом подчеркнем важное, что упор в этом комплекте (языки + пакет C4D) будет не в работе с изображениями только, но в разработке сферы применения освоенных нами программных концепций, шаблонов, парадигм. К примеру, можно спокойно создать любой вариант hardware графически, а потом писать к нему драйвер, и все в одном месте! Характер применения пакета скорее будет имитацией, нежели анимацией, но, конечно, не исключая мультипликацию. Вспомните о визуальной ограниченности, упомянутой в начале этой статьи. Огромное количество уже существующих графических библиотек и доступ к этим библиотекам из Python и С++ гарантируют нам масштабные приключения в этом виртуальном мире возможностей. Полнофункциональный пакет CINEMA 4D R15 Demo можно скачать и 42 дня работы у вас есть, пока наши умельцы не придумают к нему ключик! Если с языком Python, а начнем мы с него, возникнут проблемки, то наилучший способ решить эти проблемки - держать возле себя книгу Марк Лутц - Программирование на Python, 4-е издание, I том, 2011. Ее легко найти в сети.

**Начало.**

Мир C4D представлен иерархически, структурой типа дерево.

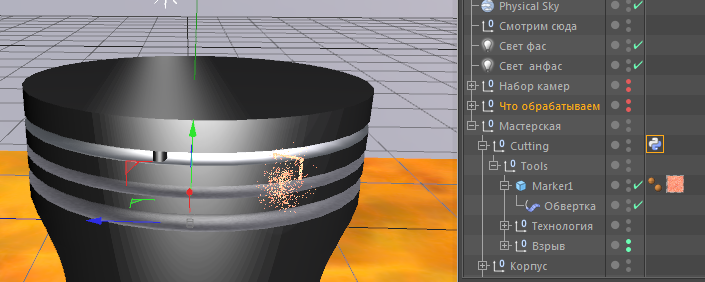


Рис.1

Пример такого дерева представлен справа на рис. 1 , это перечень всех участников нашей сцены, причем каждый участник может содержать свой набор (опять структура типа дерево) своих участников. Важна не только организация в структуру, но и последовательность элементов в списке. На изображении не учебный материал, их много в сети (YouTube, например), а вариант конкретной разработки для одной из фирм, занимающейся модернизацией двигателей (показана обработка цилиндра). Вообще, мы имеем толпу участников (объектов) сцены и у этой толпы есть только 2 способа поведения.

1. Один из них, это когда каждый идет по своей, предопределенной временем, дорожкой. Это способ работы с пакетом без программирования, когда тщательно, в каждый момент времени, для каждого объекта, задаются параметры положения и свойств с помощью инструментария TimeLine. Но если вдруг ошибка, нарушение синхронизации, определенно будут проблемы.
2. Другой же способ, заключается в том, что я назначаю управляющего этой толпой, присваивая удобному участнику сцены дополнительное свойство  , называемое тегом, с изображением логотипа Python. В этом случае я могу задать поведение каждого члена коллектива, но уже без TimeLine. Все в одном месте и под централизованным контролем. Работа с данными пользователя упрощает выполнение проекта. Отказ от TimeLine не обязателен, возможно сотрудничество.

Более того, текст может быть отлажен и удобно просмотрен в среде Eclipse. Вот фрагмент текста на рис.2:

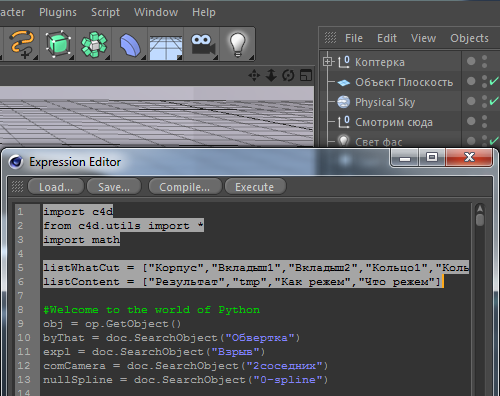


Рис. 2.

Этот фрагмент раскрывает содержимое свойства  (в дальнейшем будем называть его тегом, очень перегруженное слово в информатике!). Владелец этого тега вычисляется автоматически и внутри программы имеет предопределенное имя **op**. Если мне нужно поработать с этим владельцем, то стандартная строка такова:

obj = op.GetObject()

Для запуска отладчика (с очень большой натяжкой его можно так назвать, вся отладочная информация ваша персональная головная боль, через многочисленные print xxxxx, в Python 2.68) достаточно щелкнуть по этой иконке, сам тег выбирается и назначается так:

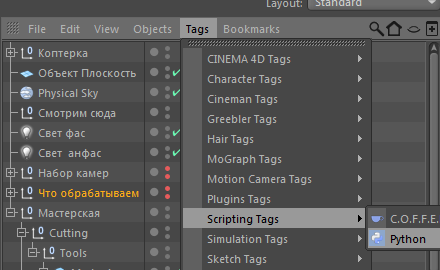


Рис.3.

Вы видите, что мы можем программировать и на языке C.O.F.F.Е. Не соблазняйтесь этой возможностью, как говорил я ранее, у нас золотым фондом возможностей являются графические библиотеки, недоступные вполне из C.O.F.F.Е. , неразумно изначально игнорировать это богатство!

Поскольку мы в самом начале, младенческом этапе программирования, то нам пока достаточно знать, как различать предметы - объекты мира C4D, как организуется запуск этого различения, как создавать новые объекты. По порядку ответы следующие:

Различение реализуется кодом:

byThat = **doc.SearchObject**("Обвертка")

В нашем дереве вы ищете объект с указанным именем ("Обвертка", имя объекта и сам объект суть разные вещи!) и находку называете удобным вам именем (byThat, имена переменных могут быть только на латинице). С этой находкой вы теперь можете делать все что допустимо, вплоть до уничтожения. Можете обратиться к свойствам находки, читать или записывать новые. Забавно отметить, что сама выполняемая программа является свойством объекта “Cutting”, см. рисунок 1. Поскольку программа может обратиться к свойствам любого объекта, то и сама к себе тоже. Сразу и не скажешь, как такую возможность использовать, но крайне любопытно попробовать (пока не пытайтесь – поведение далеко не однозначное, выключите свет, воду, выйдите из квартиры, заберите с собой домашних животных ☺ ). Обратите внимание на синтаксис **doc.SearchObject(…).**

Оказывается, что в нашей виртуальной Вселенной C4D Бюрократ (экземпляр doc класса BaseDocument) уже наследил, так как уже существует экземпляр класса c методом (перечень дозволенных действий, иначе служебные обязанности класса) **SearchObject .** Этот класс хранит все данные каждого из членов сцены.

Мы можем создать объект, любой из мира C4D. Вот пример кода:

whatCut = **c4d.BaseObject**(c4d.Onull)

В этой строке мы создаем объект Onull (Ничего). Как в реальном, физическом мире, вакуум (это некоторое Ничто из реального мира) есть просто одно из возможных состояний материи, то и в нашем мире C4D нужен объект Onull. Из этого объекта тегом  и соответствующим кодом можно создать любой другой. c4d.Onull, в примере выше, – это константа, в документации на <http://www.maxon.net/support.html> найдете перечень всех остальных. Хотя принцип образования ее предельно прост, к имени объекта дописывается символ “О” , без исключений из этого правила, обойтись не смогли.

Этот объект **c4d.BaseObject**(c4d.Onull) очень удобен для организации элементов сцены и их упорядочивания, ближайший аналог – обычный всем уже известный, банальный каталог, но с возможностью манипуляций в 4-х мерном пространстве времени и назначением свойств из preset - тегов. Все довольно интересно, этому тегу мы можем назначить текстуру анимацию, причем любую, и тогда любой объект, как вложенный подъобект будет иметь предопределенный внешний вид и поведение (вариант графического представления наследования классов!).

Мы все, наша реальная Вселенная, может быть всего то и есть результат работы подобного анимированного тега! Какой бы фильм мы не создали, но его вновь можно рассматривать как анимационный тег и применять в другом фильме, уже более высокого уровня. Но это уже вариант имитации реинкарнации!

В нашем конкретном проекте мы занимаемся модификацией поршня и его колец. Поскольку я заранее не знаю, что буду резать, то в начале наших действий я не режу ничего. Синтаксис команды вроде бы схож с ранее описанным, но есть различие – это уже конструктор объекта, не метод уже существующего, его, этого объекта, раньше не было, а мы его создали. Обидно, что можно написать **c4d.BaseObject**(c4d.Ocube) – создать куб, но аргументом конструктора не могут быть объекты, отличные от предустановленных (presets) примитивов. Если все же нужно создать задуманное, то цель достигается не копированием (явно разработчики не китайцы, обязательно было бы иначе), а модификацией свойств или разработкой новых свойств, примитивов, изменением служебных инструкций, то есть расширение классов (через инструмент plagin –ов как на C++, так и на Python).

Очень важно понимать, что время в нашей вселенной C4D строго дискретно и единица времени совпадает с обычным кадром. Как и в нашем, не виртуальном мире, вытащить время можно только через приборы, такие как часы, планшеты, так и здесь есть некоторые трудности – вот код:

baseTime = doc.GetTime()

fps = doc.GetFps()

cF = baseTime.GetFrame(fps)

Пользуясь аналогиями, а они здесь и для запоминания в том числе, мы видим, что наш Бюрократ (предопределенный класс BaseDocument с предопределеной глобальной переменной, экземпляром этого класса, и именем doc) от мира С4D является хранителем времени. По первому требованию он выдает что то похожее на часы, как прибор, а не единица времени (переменная baseTime, метод GetTime() ), затем мы узнаем частоту кадров опять же от него методом GetFps() (frame per second) и только после этого узнаем время в нашей С4D вселенной, храним это значение в переменной cF (current frame).

Если мы выбрали программный способ организации с помощью тега – Python, то осталось для каждого значения cF указать необходимые для генерации изображения действия (код). Типичная структура такова –

**if cF ==** obj[c4d.ID\_USERDATA,1]:

startDeal()

switchOn(navig("Мастерская","Вкладыш1","Что режем"),False)

switchOn(navig("Мастерская","Вкладыш2","Что режем"),False)

switchOn(expl,False)

Значение выражения obj[c4d.ID\_USERDATA,1] (obj – это хозяин нашего Python тега) определяет некоторое сF параметрически (это не константа и в любой c4d-момент может быть изменена) и далее следует перечень действий, пока не важно каких, хотя догадаться не трудно. Параметрическое задание реализуется через данные пользователя. Квадратные скобки означают структуру хранения данных, это не массив, скорее словарь.

Если для группы кадров действия одни и те же, то вместо анализа на равенство == работаем с неравенством cF > a1 and cF < a2 , где в интервале кадров (a1,a2) действия управляются одним и тем же кодом.

И последнее, нужно понимать, что код программы (результат компиляции текста) предназначен для генерации каждого кадра в отдельности, этот код выполняется столько раз, сколько кадров в нашем фильме. Более всего ситуация похожа на управление в реальном режиме времени или можно смотреть на программу как преобразователь потока кадров, предустановленных примитивов, звуковых файлов на входе в поток воспроизведения на выходе (это мы уже говорим о модели взаимодействия с кодом, опять модель вычисления, мы вышли за рамки конкретного языка).

Если же необходима разовая обработка, не кадров, а исходных данных и только в начале, то применяется вариант:



С автоматически сгенерированной заготовкой программы:

import c4d

#Welcome to the world of Python

def main():

return c4d.BaseObject(c4d.Ocube)

Код должен в обязательном порядке возвращать некоторый объект, в данном случае куб.

Остается дописать недостающие строки, отладку для достаточно большой программы (сотни строк) лучше вести в Eclipse. Выше мы видели, что мир C4D уже должен быть заселен объектами, оказывается, достаточно хотя бы одного из них, причем любого из примитивов, например Null, все остальное может быть создано программно и сами объекты (все!) и их поведение.

Но Python поведение можно усложнить еще в большей степени, добавив к этому генератору тег Python и тогда получим вообще странный конструкт:



Тег, как и в описанном ранее случае, тоже автоматически генерирует заготовку:

import c4d

#Welcome to the world of Python

def main():

pass #put in your code here

Во всех случаях обработкой кадров занимается та часть кода, которая описана в функции main().

С дополнительными возможностями таких конструкций необходимо разбираться. Работы и возможностей более чем достаточно! Качественное представление с высоты птичьего полета, уже есть. Карта местности изучения еще в деталях не видна, но уже и не белое пятно! В следующей статье мы познакомимся с классами, типичными для стандартной разработки.

**Рейдер Эдуард Ефимович, к.ф.-м.н, *freelance. 25 сентября 2013 г.***