Базовые методы обработки данных с использованием Python

Лекция 1. Стандарты анализа и обработки данных. Процесс обработки данных. Работа с Python.

Киреев Василий Сергеевич

к.т.н., доцент

Москва, 2024

Пирамида знаний

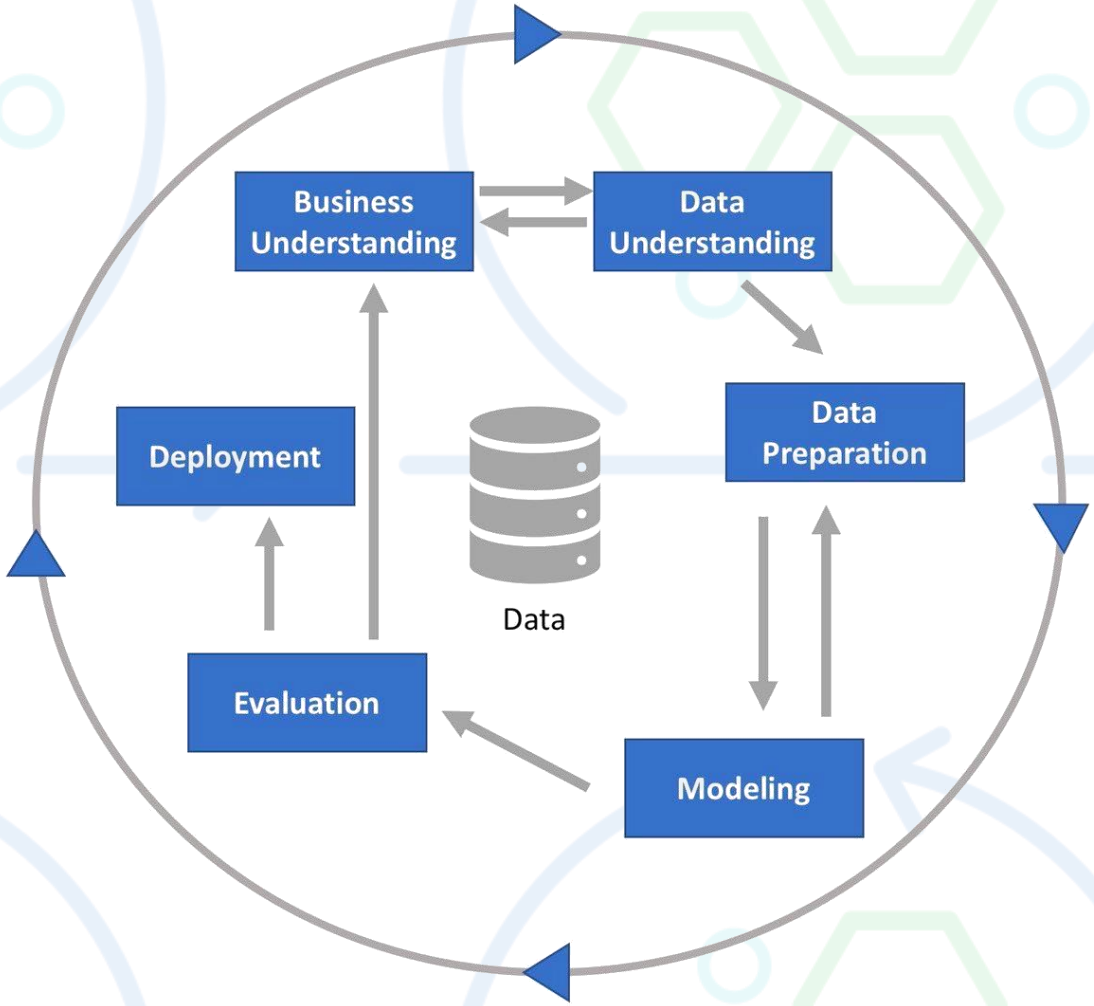
Данные VS Информация

Данные	Информация
Данные - это неорганизованные и необработанные факты	Информация - это обработанные, организованные данные, представленные в осмысленном контексте
Данные - это отдельные единицы, содержащие сырье, которое не несет никакой конкретной смысловой нагрузки.	Информация - это группа данных, которые в совокупности несут логический смысл.
Данные не зависят от информации.	Информация зависит от данных.
Одних сырых данных недостаточно для принятия решений	Информация достаточна для принятия решений
Примером данных является тестовый балл студента	Средний балл по группе студентов - это информация, полученная из приведенных данных.

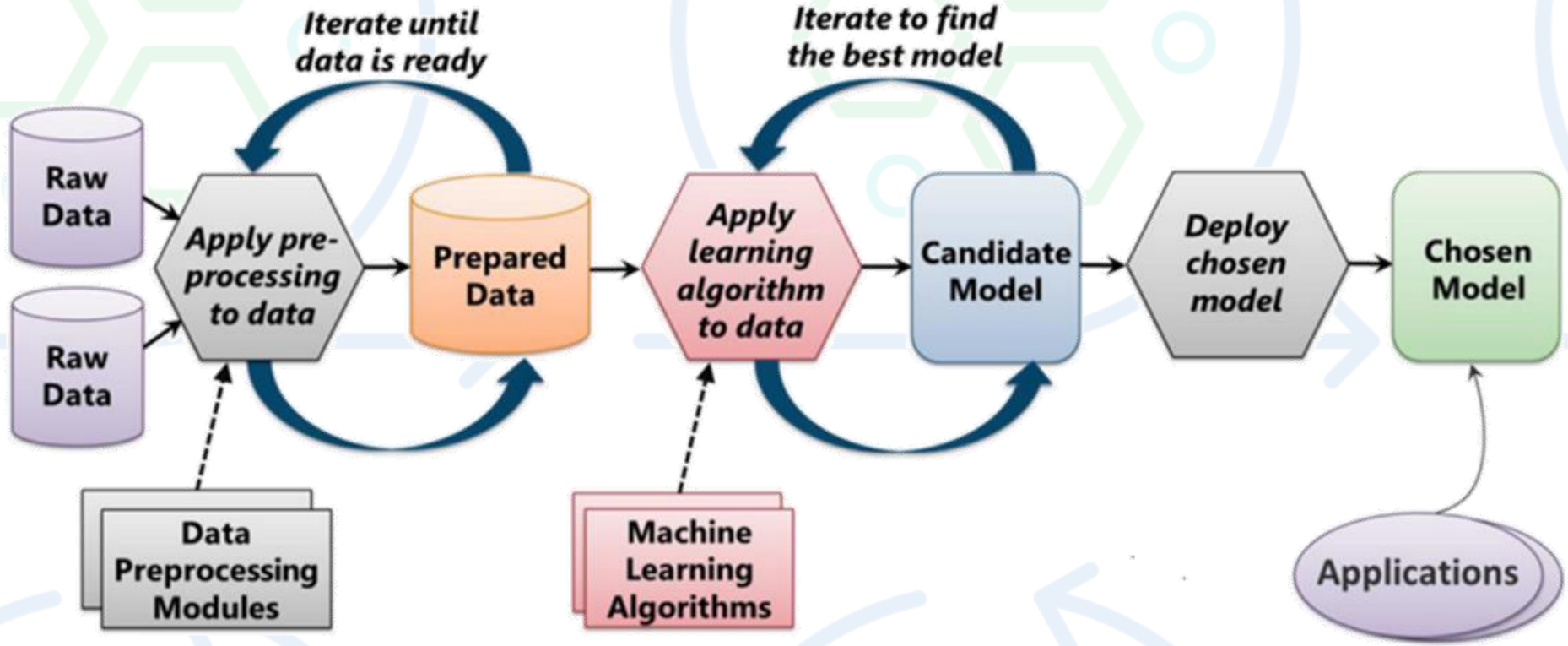
DATA-driven

Data Driven (дословно — «управляемый данными») — это подход к управлению, основанный на данных. Его главный постулат: решения нужно принимать, опираясь на анализ цифр, а не интуицию и личный опыт.

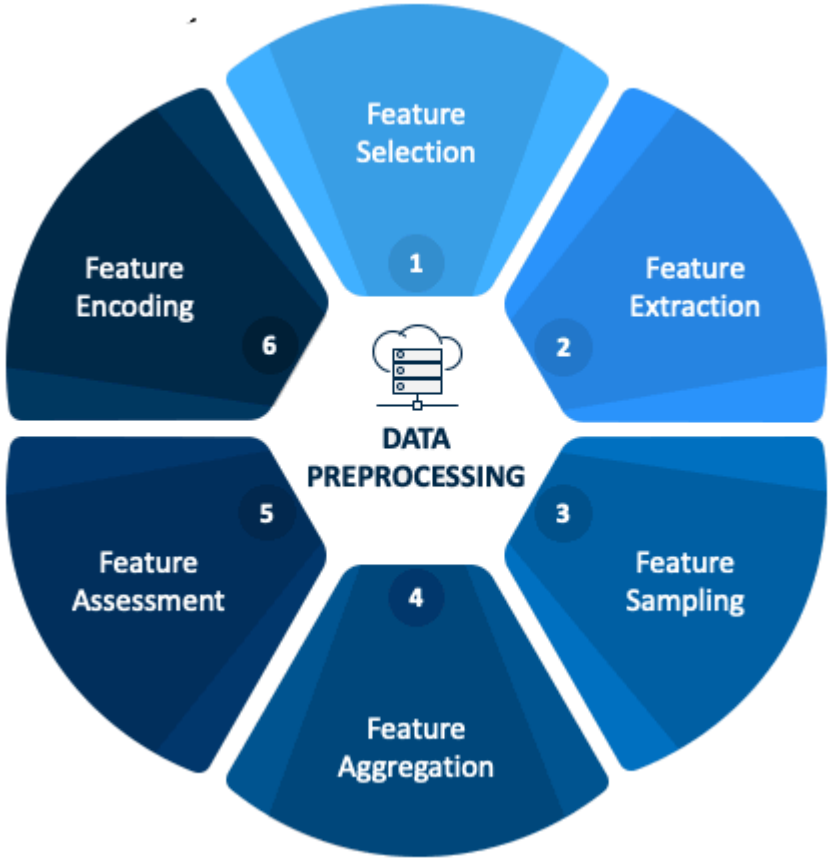
Стандарт CRISP-DM



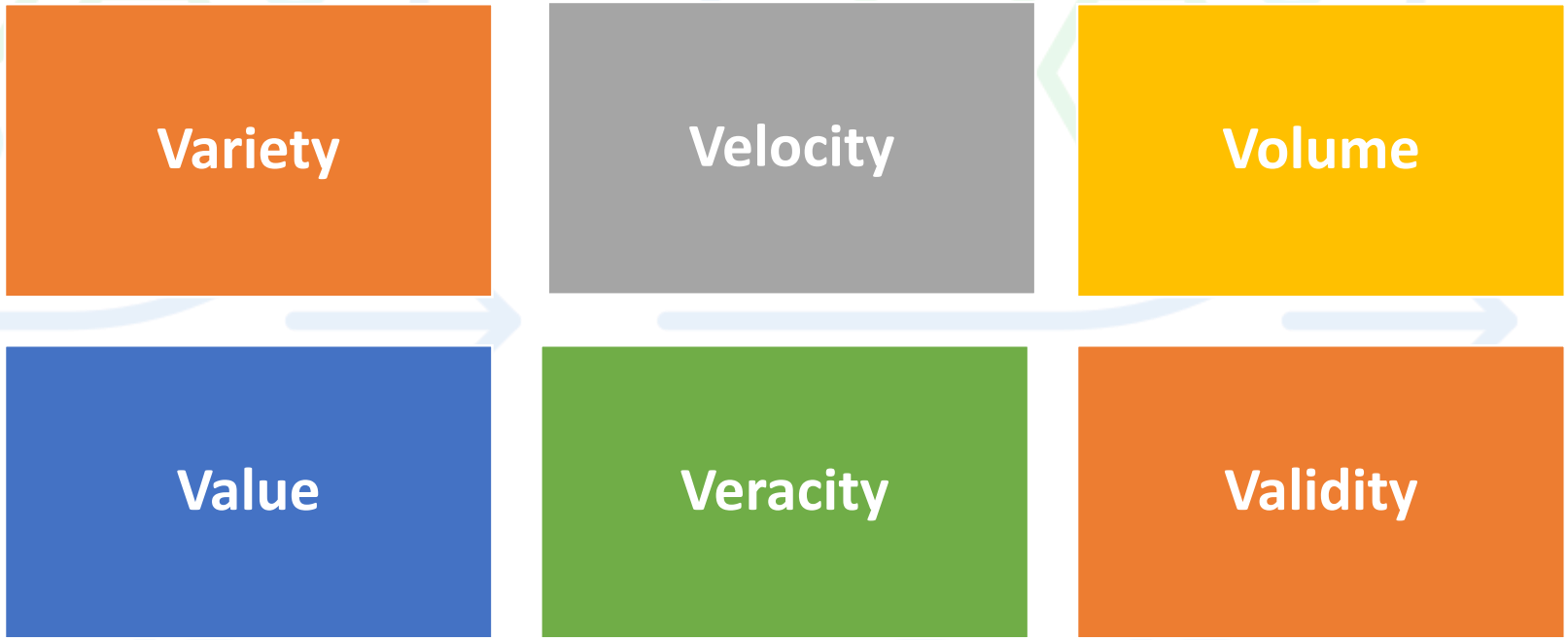
Обработка данных и её место в процессе машинного обучения



Обработка данных. Виды



V-модель Больших данных

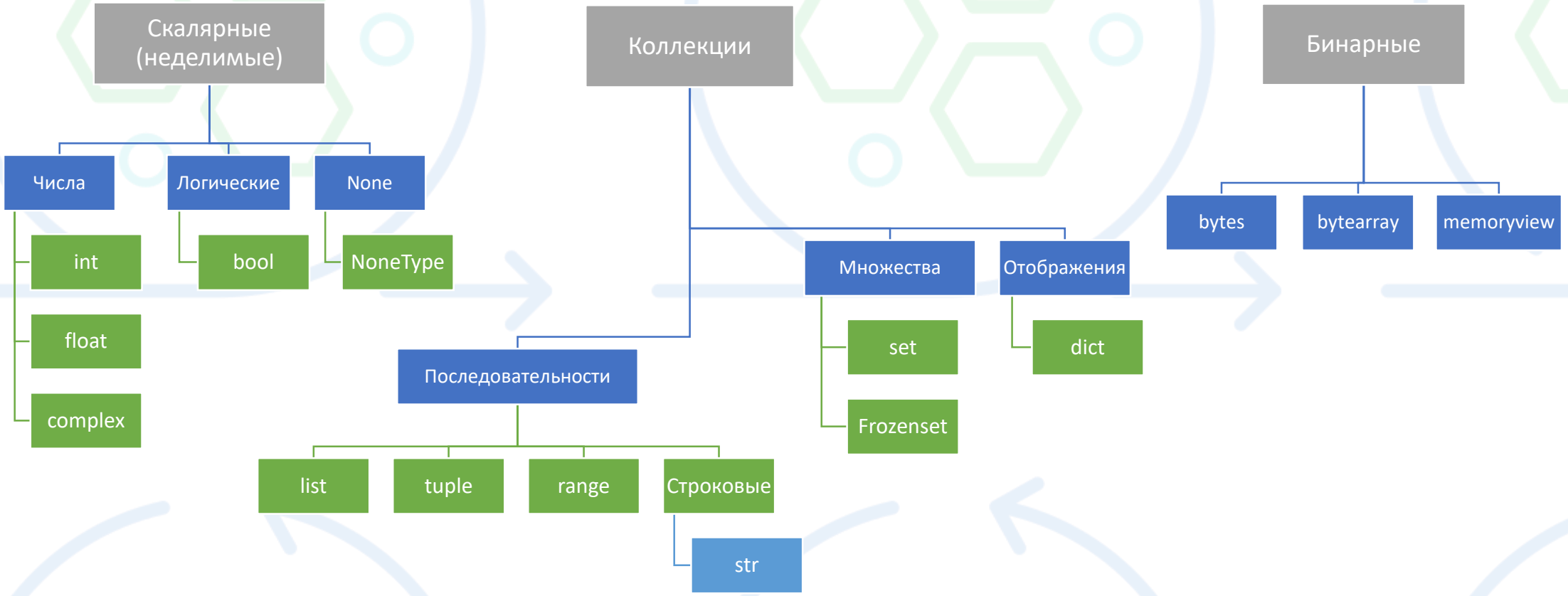


Язык Python

Python («Пайтон», «Питон») - высокоуровневый, интерпретируемый, объектно-ориентированный, императивный, строго типизированный язык общего назначения, который имеет динамическую типизацию. Разрабатывался Гвидо ван Россумом в 1989 г.. Первая публикация Python состоялась в феврале 1991 года — это была версия 0.9.0. Вторая версия Python вышла в 2000 г., третья (современная) в 2008 г.

https://www.python.org/

Встроенные типы данных в Python



Изменяемость типов данных

Изменяемые объекты	Неизменяемые объекты
Изменяемый объект может быть изменен после его создания	Неизменяемый объект не может быть изменен после его создания
Изменяемые объекты не считаются потокобезопасными по своей природе	Неизменяемые объекты считаются потокобезопасными по своей природе
Доступ к изменяемым объектам осуществляется медленнее по сравнению с неизменяемыми объектами	Доступ к неизменяемым объектам осуществляется быстрее по сравнению с изменяемыми объектами
Изменяемые объекты полезны, когда нам нужно изменить размер или содержимое нашего объекта.	Неизменяемые объекты лучше всего подходят для тех случаев, когда мы уверены, что нам не придется изменять их в любой момент времени.
Изменение изменяемых объектов является более дешевой операцией с точки зрения пространства и времени	Изменение неизменяемых объектов является дорогой операцией, поскольку при любых изменениях создается новая копия.

Изменяемые/неизменяемые типы данных в Python

Неявное преобразование типов в Python

Когда компилятор/интерпретатор какого-либо языка автоматически преобразует объект одного типа в другой, это называется автоматическим или неявным приведением. Python является сильно типизированным языком. Он не допускает автоматического приведения типов между несвязанными типами данных.

Например, строка не может быть преобразована ни в один тип чисел. Однако целое число может быть преобразовано в float. Другие языки, такие как JavaScript, являются слабо типизированными, в них целое число принудительно преобразуется в строку для конкатенации.

Неявное приведение int к float происходит при выполнении любой арифметической операции над операндами int и float.

Явное преобразование типов в Python

Хотя автоматическое или неявное приведение ограничено преобразованием int в float, можно использовать встроенные функции Python int(), float() и str() для выполнения явных преобразований, таких как преобразование строки в целое число.

Встроенная в Python функция int() преобразует целочисленный литерал в целочисленный объект, плавающее число в целое число и строку в целое число, если сама строка имеет допустимое представление в виде целочисленного литерала.

Использование функции int() с объектом int в качестве аргумента эквивалентно объявлению объекта int напрямую.

Операторы Python. if

Оператор if в Python аналогичен операторам if в других языках. Оператор if содержит булево выражение, с помощью которого сравниваются данные и на основе результата сравнения принимается решение.

Если булево выражение равно True, то выполняется блок операторов внутри оператора if. В Python утверждения в блоке равномерно отступают после символа ":". Если булево выражение равно False, то выполняется первый набор кода после конца блока.

```
discount = 0
amount = 1200
if amount > 1000:
   discount = amount * 10 / 100
print("amount = ", amount - discount)
```

```
amount = 1080.0
```

Операторы Python. for

Цикл for в Python позволяет выполнять итерации по элементам любой последовательности, например, списка, кортежа или строки. Если последовательность содержит список выражений, то он оценивается первым. Затем первый элемент (с 0-м индексом) в последовательности присваивается итерационной переменной iterating_var.

Далее выполняется блок операторов. Каждый элемент списка присваивается переменной iterating_var, и блок statement(s) выполняется до тех пор, пока не будет исчерпана вся последовательность.

```
zen = '''
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
'''
for char in zen:
  if char not in 'aeiou':
    print (char, end='')
```

```
Btfl s bttr thn gly.
Explct s bttr thn mplct.
Smpl s bttr thn cmplx.
```

Операторы Python. for-else

Python поддерживает наличие оператора "else", связанного с оператором цикла "for". Если оператор "else" используется с циклом "for", то оператор "else" выполняется, когда последовательность исчерпана до того, как управление перейдет к основной линии выполнения.

```
for count in range(6):
   print ("Iteration no. {}".format(count))
else:
   print ("for loop over. Now in else block")
print ("End of for loop")
```

```
Iteration no. 1
Iteration no. 2
Iteration no. 3
Iteration no. 4
Iteration no. 5
for loop over. Now in else block
End of for loop
```

Операторы Python. while

Оператор цикла while в языке программирования Python многократно выполняет целевой оператор до тех пор, пока заданное булево выражение истинно. За ключевым словом while следует булево выражение, а затем символ двоеточия, чтобы начать блок операторов с отступом. Здесь оператор(ы) может(ут) быть одним оператором или блоком операторов с равномерным отступом. Условие может быть любым выражением, а true - любым ненулевым значением.

```
count=0
while count<5:
   count+=1
   print ("Iteration no. {}".format(count))

print ("End of while loop")
```

```
Iteration no. 1
Iteration no. 2
Iteration no. 3
Iteration no. 4
Iteration no. 5
End of while loop
```

Операторы Python. Match-Case

Оператор match принимает выражение и сравнивает его значение с последовательными шаблонами, заданными в виде одного или нескольких блоков case. Выполняется только первый совпавший паттерн.

Также возможно извлечение компонентов (элементов последовательности или атрибутов объекта) из значения в переменные. Основное использование match-case заключается в сравнении переменной с одним или несколькими значениями.

```
def access(user):
  match user:
    case "admin" | "manager": return "Full
access"
    case "Guest": return "Limited access"
    case _: return "No access"
print (access("manager"))
```

```
Full access
```

Операторы Python. break

break завершает текущий цикл и возобновляет его выполнение со следующего оператора, как и традиционный оператор break в языке C. Наиболее часто оператор break используется при возникновении внешних условий, требующих поспешного выхода из цикла.

Оператор break может использоваться как в циклах while, так и for.При использовании вложенных циклов оператор break останавливает выполнение самого внутреннего цикла и начинает выполнение следующей строки кода после блока.

```
for letter in 'Python':
   if letter == 'h':
      break
   print ('Current Letter :', letter)
```

```
Current Letter : P
Current Letter : y
Current Letter : t
```

Операторы Python. continue

Оператор continue в Python возвращает управление в начало текущего цикла. Оператор continue может использоваться как в циклах while, так и for.

Оператор continue прямо противоположен оператору break. Он пропускает оставшиеся операторы в текущем цикле и начинает следующую итерацию.

```
num = 60
print ("Prime factors for: ", num)
d=2
while num > 1:
  if num%d==0:
    print (d)
    num=num/d
    continue
  d=d+1
```

```
Prime factors for: 60
2
2
3
5
```

Операторы Python. pass

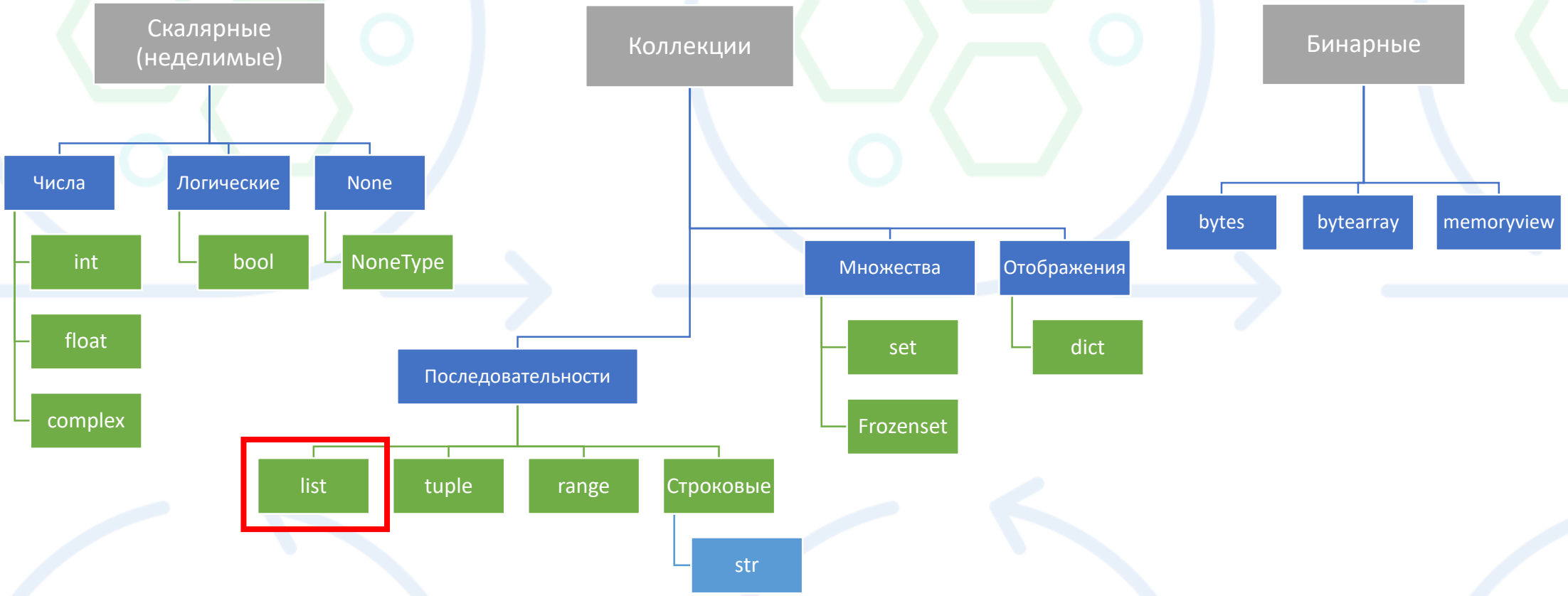
Оператор Python pass используется в тех случаях, когда синтаксически требуется выполнение оператора, но при этом не требуется выполнение какой-либо команды или кода.

Оператор Python pass является нулевой операцией; при его выполнении ничего не происходит. Оператор pass также полезен в заглушках.

```
for letter in 'Python':
   if letter == ‘y':
      pass
      print (‘pass')
   print (‘Буква:', letter)
print («Конец")
```

```
Буква: P
Буква: y
pass
Буква: t
Буква: h
Буква: o
Буква: n
Конец
```

Встроенные типы данных в Python



Списки в Python

Список - это один из встроенных типов данных в Python. Список Python - это последовательность элементов, разделенных запятыми и заключенных в квадратные скобки [ ]. Элементы списка Python не обязательно должны иметь одинаковый тип данных.

Список в Python аналогичен массиву в C, C++ или Java. Однако основное отличие заключается в том, что в C/C++/Java элементы массива должны быть одного типа. С другой стороны, в списках Python могут находиться объекты с разными типами данных.

Список Python является изменяемым. Любой элемент списка может быть доступен по его индексу и может быть изменен. Один или несколько объектов из списка могут быть удалены или добавлены. Список может содержать один и тот же элемент более чем в одной позиции индекса.

Списки. Операции

Операция	Описание
x in l	true, если элемент x есть в списке l
x not in l	true, если элемент x отсутствует в l
l1 + l2	объединение двух списков
l * n , n * l	копирует список n раз
len(l)	количество элементов в l
min(l)	наименьший элемент
max(l)	наибольший элемент
sum(l)	сумма чисел списка
for i in list()	перебирает элементы слева направо

Списки. Генераторы списков

Генератор списков (list comprehension) - очень мощный инструмент программирования. Он похож на нотацию построителя множеств в математике. Это лаконичный способ создания нового списка путем выполнения некоторого процесса над каждым элементом существующего списка. Генератор списков значительно быстрее, чем обработка списка с помощью цикла for.

```
list1=['a', 3]
list2=[3, 6.7]
Lst12=[(x,y) for x in list1 for y in list2]
print (Lst12)
```

```
[('a', 3), ('a', 6.7), (3, 3), (3, 6.7)]
```

Кортежи

В языке Python кортеж (tuple) - это тип данных последовательности. Он представляет собой упорядоченную коллекцию элементов. Каждый элемент кортежа имеет уникальный позиционный индекс, начиная с 0. В массиве C/C++/Java элементы массива должны быть одного типа. С другой стороны, кортеж в Python может содержать объекты разных типов данных.

И кортеж, и список в Python являются последовательностями. Одно из основных различий между ними заключается в том, что список в Python является изменен, а кортеж - неизменяемым. Хотя к любому элементу из кортежа можно обратиться, используя его индекс, он не может быть изменен, удален или добавлен.

Строки

В Python строки - это упорядоченная последовательность символов Unicode, , используемые для хранения и представления текстовой информации, поэтому с помощью строк можно работать со всем, что может быть представлено в текстовой форме.

Каждый символ в строке имеет уникальный индекс в последовательности. Индекс начинается с 0. Первый символ в строке имеет позиционный индекс 0. Индекс продолжает увеличиваться по направлению к концу строки.

Строки. Экранирование

Экранированная последовательность	**Назначение**
\n	Перевод строки
\a	Звонок
\b	Забой
\f	Перевод страницы
\r	Возврат каретки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\N{id}	Идентификатор ID базы данных Юникода
\uhhhh	16-битовый символ Юникода в 16-ричном представлении
\Uhhhh…	32-битовый символ Юникода в 32-ричном представлении
\xhh	16-ричное значение символа
\ooo	8-ричное значение символа
\0	Символ Null (не является признаком конца строки)

Если перед открывающей кавычкой стоит символ 'r' (в любом регистре), то механизм экранирования отключается.

Строки. Методы

Методы	Описание
S = 'str'; S = "str"; S = '''str'''; S = """str"""	Литералы строк
S = "s\np\ta\nbbb"	Экранированные последовательности
S = r"C:\temp\new"	Неформатированные строки (подавляют экранирование)
S = b"byte"	Строка байтов
S1 + S2	Конкатенация (сложение строк)
S1 * 3	Повторение строки
S[i]	Обращение по индексу
S[i:j:step]	Извлечение среза
len(S)	Длина строки
S.find(str, [start],[end])	Поиск подстроки в строке. Возвращает номер первого вхождения или -1
S.rfind(str, [start],[end])	Поиск подстроки в строке. Возвращает номер последнего вхождения или -1

Строки. Форматирование

Форматирование строк - это процесс динамического построения строкового представления путем вставки значений числовых выражений в уже существующую строку.

Оператор конкатенации строк в Python не принимает операнд, не являющийся строкой. Поэтому Python предлагает следующие приемы форматирования строк

- Использование оператора % для подстановки
- Использование метода format() класса str
- Использование синтаксиса f-строк
- Использование класса String Template

Строки. Форматирование. Оператор %

Одной из самых замечательных возможностей Python является оператор формата строк %. Этот оператор уникален для строк и компенсирует отсутствие функций из семейства printf() языка C. Символы спецификации формата (%d %c %f %s и т.д.), используемые в языке C, используются в строке в качестве заполнителей.

```
имя=" Rajesh "
возраст=30
print ("Меня зовут %s, а мой возраст
составляет %d лет" % (имя, возраст))
```

```
Меня зовут Rajesh, а мой возраст составляет
30 лет
```

Строки. Форматирование. Метод f-строк

В версии 3.6 в Python появился новый метод форматирования строк - f-strings или Literal String Interpolation. С помощью этого метода форматирования можно использовать встроенные выражения Python внутри строковых констант.

Python f-строки являются более быстрыми, читабельными, лаконичными и менее подверженными ошибкам. Строка начинается с префикса 'f', в нее вставляется один или несколько держателей, значение которых заполняется динамически.

```
имя = 'Rajesh'
возраст = 30
fstring = f'Меня зовут {имя} и мне {возраст}
лет'
print (fstring)
```

```
Меня зовут Rajesh и мне 30 лет
```

Множества

Множество set в python - "контейнер", содержащий не повторяющиеся элементы в случайном порядке. Отличие set от frozenset заключается в том, что set - изменяемый тип данных, а frozenset - нет.

Преимущества:

- Дают возможность быстро удалять дубликаты, поскольку, по определению, могут содержать только уникальные элементы;
- Позволяют, в отличие от других коллекций, выполнять над собой ряд математических операций, таких как объединение, пересечение и разность множеств.

Словари

Словарь - один из встроенных типов данных в Python. Словарь в Python является примером типа отображения. Объект отображения "сопоставляет" значение одного объекта с другим.В языковом словаре есть пары слово и соответствующее значение.

Две части пары - ключ (слово) и значение (смысл). Аналогично, словарь Python также представляет собой набор пар ключ:значение.

Пары разделяются запятой и помещаются в фигурные скобки {}.Чтобы установить соответствие между ключом и значением, между ними ставится символ двоеточия ':'.

Словари. Методы

Методы	Описание
dict.clear()	очищает словарь.
dict.copy()	возвращает копию словаря.
classmethod dict.fromkeys(seq[, value])	создает словарь с ключами из seq и значением value (по умолчанию None).
dict.get(key[, default])	возвращает значение ключа, но если его нет, не бросает исключение, а возвращает default (по умолчанию None).
dict.items()	возвращает пары (ключ, значение).
dict.keys()	возвращает ключи в словаре.
dict.pop(key[, default])	удаляет ключ и возвращает значение. Если ключа нет, возвращает default (по умолчанию бросает исключение).
dict.popitem()	удаляет и возвращает пару (ключ, значение). Если словарь пуст, бросает исключение KeyError. Помните, что словари неупорядочены.

Словари. Доступ к элементам

Словарь в Python не является последовательностью, так как элементы в нем не индексируются. Тем не менее, для получения значения, связанного с определенным ключом в объекте словаря, можно использовать оператор квадратных скобок "[ ]".

Python выдает ошибку KeyError, если ключ, указанный внутри квадратных скобок, отсутствует в объекте словаря. Метод get() класса dict в Python возвращает значение, сопоставленное с заданным ключом.

```
capitals = {"Россия": "Москва",
"Великобритания": "Лондон"}
print ("Столица Великобритании - это: ",
capitals.get('Великобритания'))
print ("Столицей России является: ",
capitals['Россия'])
```

```
Столица Великобритании - это: Лондон
Столицей России является: Москва
```

Функции Python

Функция в Python - объект, принимающий аргументы и возвращающий значение. Обычно функция определяется с помощью инструкции def. Функция может принимать произвольное количество аргументов или не принимать их вовсе. Также распространены с позиционными и именованными аргументами, обязательными и необязательными.

Функция может быть любой сложности и возвращать любые объекты (списки, кортежи, и другие функции). Одной из основных причин выделения функций из остального кода является необходимость в их многократном использовании.

Функции Python. Виды

Стандартная библиотека Python включает в себя множество встроенных функций. К числу встроенных функций Python относятся print(), int(), len(), sum() и др. Эти функции всегда доступны, так как загружаются в память компьютера сразу после запуска интерпретатора Python.

В стандартную библиотеку также входит ряд модулей. Каждый модуль определяет группу функций. Эти функции не являются легкодоступными. Их необходимо импортировать в память из соответствующих модулей.

Помимо встроенных функций и функций встроенных модулей, можно создавать собственные функции. Такие функции называются пользовательскими.

Функции Python. Виды аргументов

Позиционные или обязательные аргументы

Аргументы с ключевыми словами

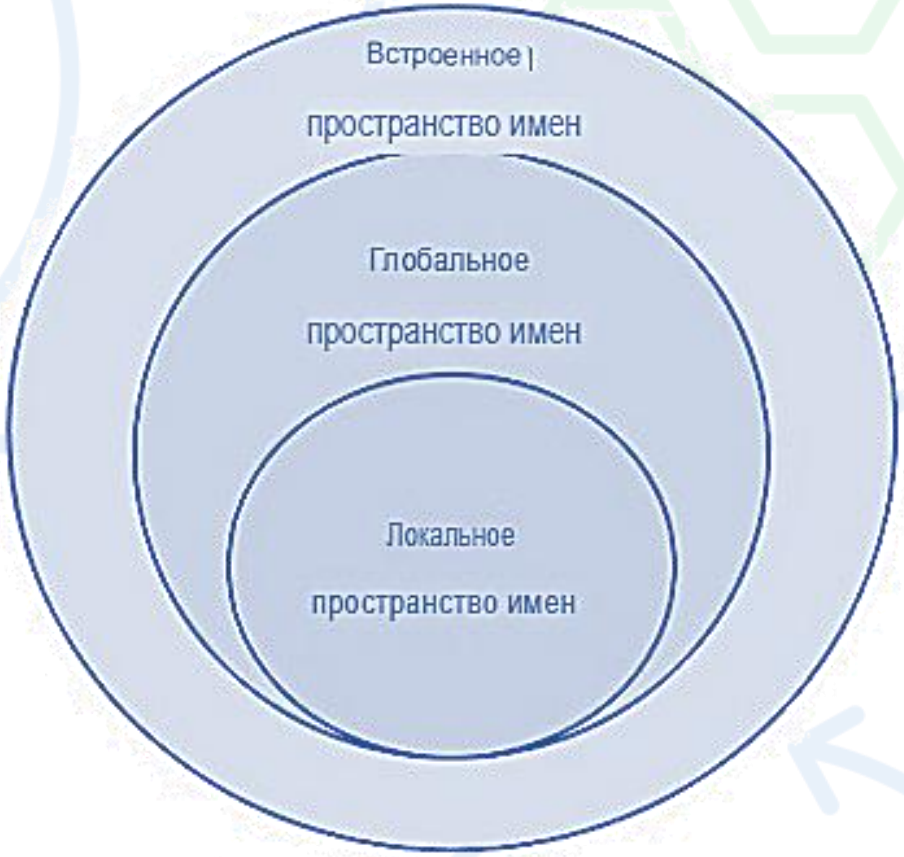
Аргументы по умолчанию

Только позиционные аргументы

Аргументы только для ключевых слов

Аргументы произвольной или переменной длины

Функции Python. Пространство имен



Тип пространств имен

Аннотация функции в Python

Аннотация функции в Python позволяет добавлять дополнительные поясняющие метаданные об аргументах, объявленных в определении функции, а также о возвращаемом типе данных.

Хотя для документирования функции в Python можно использовать функцию docstring, она может устареть при внесении определенных изменений в прототип функции. Поэтому в Python в результате выпуска PEP 3107 была введена функция аннотаций.

Аннотации не учитываются интерпретатором Python при выполнении функции. В основном они предназначены для IDE Python, чтобы предоставлять программисту подробную документацию.

Аннотации - это любые допустимые выражения Python, добавляемые к аргументам или возвращаемому типу данных. Простейшим примером аннотации является указание типа данных аргументов. Аннотация указывается как выражение после двоеточия перед аргументом.

Анонимные выражения в Python

Lambda-функция позволяет определять функцию анонимно. Стоит отметить, что она является именно функцией, а не оператором. То есть лямбда-функция возвращает значение, и у нее есть неявный оператор return. Лямбда принимает любое количество аргументов (или ни одного), но состоит из одного выражения. Лямбда-выражения в Python и других языках программирования имеют свои корни в лямбда-исчислении, модели вычислений, изобретенной Алонзо Черчем (Alonzo Church).

Применение lambda-функций

```
1 y = lambda x: x+1
2 y(4)
```

```
5
```

```
1 y = lambda x: print('Hey!')
2 y(3)
```

```
Hey!
```

```
1 y = lambda: print('Hey!')
2 y()
```

```
Hey!
```

```
1 func = lambda *args: args
2 func('a','b',10)
```

```
('a', 'b', 10)
```

Спасибо за внимание!