Информатика

Лекция 1

Осенний семестр 2023

Организационные вопросы

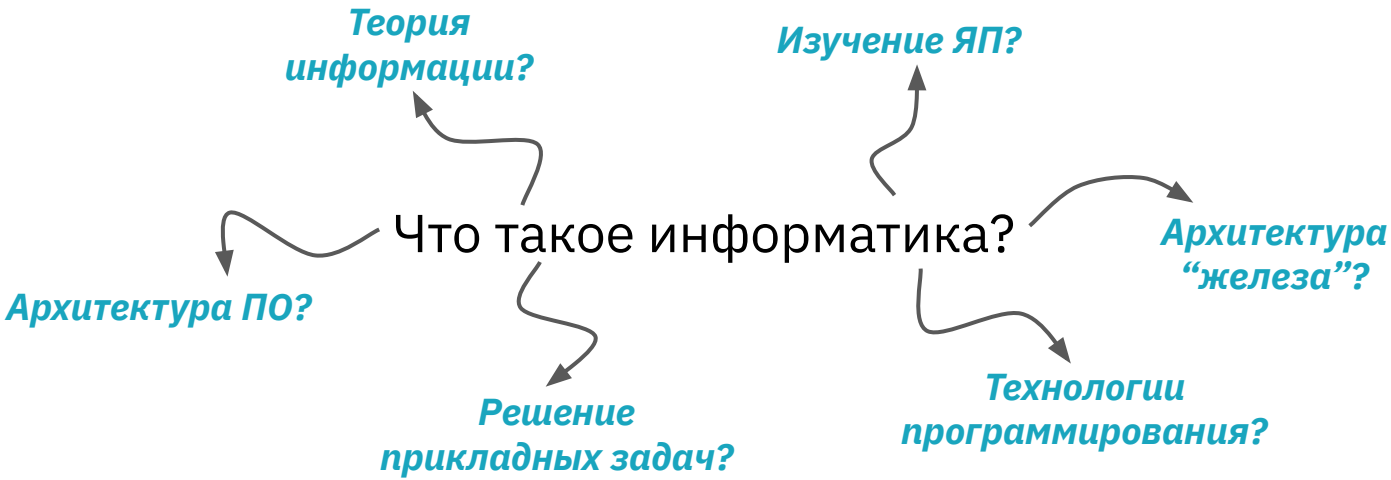
Знакомимся

Гольдштейн Клим Дмитриевич (Клим, ты/вы – как удобно)
ФЭФМ’22, аспирант
м.н.с. лаборатории многомасштабного моделирования в физике мягкой материи
ML-инженер в Softline
ст. преподаватель ФЭФМ и ФПМИ

По каким вопросам обращаться:

- Вопросы по материалу лекций
- Вопросы по наполнению курса
- Консультация по стажировкам в лабораториях/компаниях
- Проблемы с контестами
- Спорные ситуации по оцениванию

Цель и особенности курса



Наши цели:

1) Обрести практические навыки программирования, необходимые для решения научных и повседневных задач
2) Получить универсальные теоретические знания, достаточные для профессионального общения и дальнейшей специализации в любой из областей IT
3) Приобрести набор умений, достаточный для трудоустройства в топовую IT-компанию или лабораторию, связанную с IT

План на семестр

Первая половина – практика программирования

На лекциях – особенности языка, теория программирования
Примеры заданий на семинарах:

1) Обработка лабораторной работы
2) Построение графиков, работа с БД
3) Визуализация модели Солнечной системы.
4) Написание простой игры

Вторая половина – введение в алгоритмы и структуры данных

На лекциях – теория по алгоритмам
На семинарах – решение алгоритмических задач (как теоретических, так и с реализацией в коде), набор задач зависит от уровня группы

Возможные сложности

1) Где искать информацию, если упустил что-то из теории ʕ ͡° ͜ʖ ͡°ʔ
   a) **Just google it.** Лучше на английском. На 95% вопросов есть ответы на stackoverflow, github, habr, в документации языка/библиотек и других подобных ресурсах. Лучше сразу смотреть ответ на вопрос в нескольких источниках.
   b) Основная книга по алгоритмам: Т.Кормен “Алгоритмы: построение и анализ”. Книги по синтаксису языка лучше не читать, а искать статьи по конкретным темам.
2) У меня ошибка в коде, я не понимаю, что не так **(ノಠ益ಠ)ノ彡┻━┻**
   a) **Загуглите** текст ошибки. Переберите все возможные крайние случаи, которые могли возникнуть в вашем коде.
   b) Спросите соседа/одногруппника/однокурсника, чтобы он посмотрел на ваш код свежим взглядом.
   c) Спросите семинариста. Спросите в чате курса.
3) Я ничего не понимаю и не могу написать ни строчки кода **(ಥ_ಥ)**
   a) **Все с этого начинали.** Попробуйте представить, как бы вы решали задачу “вручную” и написать код, соответствующий этим действиям.
   b) Попросите знакомого объяснить вам тему и **фиксируйте логику рассуждений.** Постарайтесь понять, как разбивать задачи на подзадачи и как подходить к решению подзадач и “сборке” итогового решения. **Не пишите то, чего не понимаете.**

Система оценивания

$$G = 0.6*S + 0.4*C + E$$

G – итоговая оценка за семестр

S – накопленная оценка за семестр (до 10)
5 баллов – выполнение лабораторных работ на семинарах
5 баллов – выполнение контестов (домашних заданий)
1 балл – лекционные “разогревочные тесты”

C – оценка за зачет (до 10)
Зачет сдается другому семинарскому преподавателю на зачетной неделе в формате теоретического опроса по программе курса.

E – бонус от семинариста (до 1)

Блокирующие оценки
Для допуска к зачету необходимо набрать минимум 2 балла за лабораторные и 2 балла за контесты.

Списывание жестко наказывается
Контесты после дедлайна будут анализироваться в системе поиска плагиата. При поимке на
1 задаче – аннулируется задача (у обоих).
2 задачах – аннулируется контест.
3 задачах – недопуск к зачету.

Использование LLM (ChatGPT/GPT-4, LLaMA 2 etc.)

Для чего имеет смысл и можно использовать:

1) Получение кратких (но не исчерпывающих) справок по каким-либо темам. Пример: “What’s the difference in memory management in C++ and Python?”
2) Помощь в обработке ошибок. Пример: “I’ve got the “division by zero” error in my Python code. How can I fix it?”

Использовать с осторожностью:

1) Перенос/корректирование кода. С очень высокой вероятностью код будет с ошибками или неэффективный. Чтобы это выявить, нужно уметь разбираться в коде и алгоритмах.
2) Написание кода в малознакомых библиотеках. Например: “Write a code to cut the DataFrame by date in Pandas”

Не использовать:

1) Написание кода с нуля. Сначала научитесь сами, иначе не сможете понять, где можно сделать лучше.
2) Решение алгоритмических и математических задач. Порой искать ошибки в сгенерированных решениях сложнее, чем решить задачу самому.

Общий принцип: **Не пишите то, чего не понимаете.**

Группы и потоки

Канал курса – https://t.me/+diulQr_bGsM1YWEy, для уведомлений, рассылки материалов
Чат курса – https://t.me/+h-y9On7yZBExYTli, для обсуждения задач и вопросов студентов

3.5 уровня – начальный (Кулиев, Дивари, Корнев), основной (Сагирова, Воронов), продвинутый (Юдаев, Селезнев) и проектный (Гольдштейн, Коган)

Распределительный контест на первом семинарском занятии, в проектную группу – дополнительное интервью

Перевестись между группами (кроме проектной) можно по согласованию с отпускающим и принимающим преподавателем

Лекция 1

Основы Python

Почему Python?

Aug 2023	Aug 2022	Change	Programming Language	Ratings
1	1		Python	13.33%
2	2		C	11.41%
3	4	▲	C++	10.63%
4	3	▼	Java	10.33%
5	5		C#	7.04%

1) Простота освоения
2) Актуальность/популярность
3) Наличие большого количества прикладных библиотек

Почему не только Python?

Дает мало контроля пользователю – не все концепции программирования можно изучить

Плох для высоконагруженных вычислений

Диверсификация: разные языки для разного времени и задач

Total					
	Energy		Time		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84

Данные в Python

В Python **ссылочная модель данных** (об этом еще будем говорить) и **сильная динамическая типизация**

Неизменяемые типы данных

- Числовые (int, float);
- Логический (bool)
- Строки (str) — нельзя менять отдельные буквы в строке — только создав новую строку;
- Кортеж (tuple) — не позволяет изменять набор, но может содержать изменяемые элементы;
- Замороженное множество (frozenset);

Изменяемые типы данных

- Список (list) — последовательность любых элементов
- Множество (set) — неповторяющийся набор *неизменяемых* элементов
- Словарь (dict) — таблица соответствия *ключ → значение*. *Ключ* обязательно *неизменяемый*, *значение* любое.

Для определения типа переменной есть функция `type`. Для проверки типа — `is`. `A = type(x) is int`

Арифметические операции

Первый приоритет – возведение в степень X = Y**2 (X = Y**0.5 – корень), выполняется справа налево

Второй приоритет – унарный минус

Третий приоритет – умножение и деление, имеют одинаковый приоритет и выполняются слева направо (a/b*c = (a/b)*c)
Целочисленное деление – a//b – деление нацело, a%b – остаток (как быть с отрицательными числами?)

Составное присваивание a += b (также -=, *=, /=, %=)

Каскадное присваивание `a = b = c = 0`
Множественное (кортежное) присваивание `a, b = 1, 2`

Циклы и ветвления

Цикл `for` позволяет "пробегать" по всем элементам списка или генератора.

```
for <variable_name> in <data>:
  '''
  some code
    '''
```

Вывод элементов списка

```
for i in [123, "string", (1,2,3)]:
    print(i)
```

Пример вывода *целочисленной* арифметической прогрессии:

```
start = 1   # первый элемент прогрессии
stop = 100 # ограничение прогрессии
step = 2   # шаг прогрессии

for i in range(start, stop, step):
    print(i)
```

Цикл while

```
while <условие продолжения цикла>:
  '''
  some code
    '''


else: # выполняется после выхода из цикла, может отсутствовать


'''
  some code
    '''
```

Пример вывода *целочисленной* арифметической прогрессии:

```
start = 1   # первый элемент прогрессии
stop = 100 # ограничение прогрессии
step = 2   # шаг прогрессии


i = start
while i < stop:
  print(i)
    i += step
```

Ветвление

Общий синтаксис:

```
if <условие 1>:
  '''
  выполняется, если <условие 1> истинно
  '''
elif <условие 2>:
  '''
  выполняется, если <условие 2> истинно, а предыдущие ложны. Может отсутствовать.
  '''
elif <условие 3>:
  '''
  выполняется, если <условие 3> истинно, а предыдущие ложны. Может отсутствовать.
  '''
.....
else:
  '''
  выполняется, если все условия ложны. Может отсутствовать.
    '''
```

Логическое сравнение – `==, !=, >, <, >=, <=. not` меняет логическое значение на противоположное

Для нескольких условий – `and, or`

Управление циклом

- `break` — для преждевременной остановки цикла. Выход происходит только из одного цикла (в котором написана команда)
- `continue` — для перехода к следующей итерации текущего цикла

Пример использования:

```
i = 1
while True:     # запускаем бесконечный цикл
  i += 1        # увеличиваем i
  if i > 100:
    break       # останавливаемся, если перешли через 100
  if i % 2 == 0:
    continue    # пропускаем, если число чётное

    print(i)      # печатаем, если сюда
```

Интересные ссылки

https://benchmarksgame-team.pages.debian.net/benchmarksgame/index.html

https://www.sciencedirect.com/science/article/pii/S0167642321000022