Все конспекты велись в терминале и просто преобразованы из его текста с пояснениями. Вообще проверять какие-то фишки питона в отрыве от решения задачек и загрузки их в системы по типу ежужа в терминале очень удобно:

1. Язык интерпретируемый (нет отдельного момента компиляции в исполняемый файл и запуска оного; интерпретатор просто построчно читает твой код и построчно его исполняет), благодаря чему можно просто запустить сессию питончика в терминале и писать построчно код, как одну прогу - все переменные запомнятся, всё будет работать итд
2. Так как каждая команда как бы самостоятельно отрабатывает, если ты где-то ошибся, тебе сразу выбросит исключение\ошибку и ты просто исправляешь логический блок, в котором совершил фигню (логический блок == если где-то под конец цикла налажал, придётся переписать весь цикл, но, блинб, зато скачивать не надо ничего)
3. Существует 100500 интерпретаторов питончика в терминал, с подсцветками синтаксиса, с подсказками команд, можно найти что угодно под себя. У меня будет преимущественно использоваться просто python3.11 (сейчас, кста, уже есть 3.12, и лучше качать его и пользоваться им, просто держим в голове, что все команды, которые я писал тут, лучше падаванить {падаванить == самому протыкать пальчиками вместе со мной}, чтобы видеть реакцию питончика, вдруг в новой версии что-то поменялось [хеши починили, хехе, но до этого мы дойдём в 8ой главе]) или ptpython. Устанавливаются они просто базовым ***sudo apt install <>***

Итак, с терминалом разобрались, погнали про объекты. Объекты у нас здесь есть почти наверное любых типов, будем потихоньку про все говорить и смотреть на них

Сначала просто приятный родной вывод, нам всё таки как-то для общения надо что-то печатать))) используется функция ***print,*** в которой через запятую перечисляются параметры, которые будут выводиться. Параметры автоматически разделяются пробелами, каждый вывод заканчивается переносом строки, если не сказано иное (мысль “если не сказано иное” будет много где, потому что очень много функций питона имеют огромное количество параметров со значениями по умолчанию. Так, н-р, print имеет параметры разделителя аргументов ***sep=’ ‘***, окончания строки ***end=’\n’***)

Ещё одна фишка терминала: т.к. любой объект это экземпляр своего класса, а любаяоперация - метод этого класса. вызываемый экземпляром, любое действие имеет как бы своё возвращаемое значение (ну, логично, в целом, всё на функциях и методах). И терминал умеет это самое возвращаемое значение, если оно никуда не присваивается, просто выбрасывать на вывод. Так что для вот этой самой проверочной работы себя в терминальчике можно даже print не писать. Другой вопрос, что иногда то, что идёт в print, и то, что идёт возвращаемым значением, это разные вещи. Но потом на практике станет понятно

stephen@The-Night-Road:~$ ptpython # тот самый ptpython - одна из программ питончика в терминале
>>> print("Beauty\n"*5)
Beauty

Beauty
Beauty
Beauty
Beauty

Итак, объекты. В питоне есть числа - целочисленные и с плавающей точкой. При этом ограничения на размер этих чисел у нас нет от слова совсем.

```
>>> 1 + 5
6
>>> 12345 ** 789
153629061464580456266468227559093134330026464661424787504450783009551411081875805919164119020835302706290512181710540320531090967810349852235188150439039450878150781500959352210984935322994983466130373089555017428405602507072689027678371427233829553141336481265157994092946386025220538269453499541346735415221797515652278097643249486925254632907728592391567216183315318381297521510616494950230503977755350633250188926213619925749672788743732087670514141442766358452582722087870218827950527738117234803449768350659319716813881077481486639202742951408358698010358618502740319446378295932361741493105302202334210506204432342264078032994174743761266161232582054794072252209220182306647276827038144934918306837331032072109147973734955837198300353297601805894058182987829193344592790396055174800795650493844104404359817116304802503285455446662984893621828580326667434328206225081454182732444174840611617986896007530939520751011560533898579240280833519745375243735929352366187865128738933174456696739659220461367023073657395017809885334149178176211657618671416824628054870719732219022612730274379828233266347401369591453773680533745156958264097447724753288625948101365234660207323111237253508929488255808731468525938079661330681782896342525579396488582145047100399675153713597415209501341174215886906341989025416488185618632391941462581447613358342624810687818943152559325296870987401689784668347906716966890026449982856059420072243754030025975667109428561320166001478234067475253891218322231377329456857651695175398037773901406037384419502493596967036402531857205408719966329387029986013286088546558862698418144543800303980311975988538086344348275437685835640977735961929498677055335812747277164791983276074577524781828997554426024125090438032854623367295174437247498297545593819638283977252355225972256254989950962759568854868655020950137357507386209834650134147728239529755176617959292080945597237705570218518153013523466435057491335788895698483507784308290046498206336106785786289312236147691226744630326187594193860826516866545355572759661813510396841069249520922699552897172535072560839772277110434976125249606794340235298038877658499953952210326422112799588201560532412333384560228946852996902670786505961465641799065173327816423920928790030594092558898677592756545676835704155938802747236208460064080644854656148072276845418883438587874565553149568865152316245698469426517659752506593071007117508977252534361786606433417924934019421533769740712402082115384963754584823374133943205859910537584687522116915831528356329804688546189118859526762818466673073266501207377160820981038235053975566883283265386686208933401564532123647417
```

4157502200404168898328617212901182848002329681876766070084015518260935657
4950010843484048618316291240464228750148978395611765587946097310359707834
5922259634964149895695926500349995003184394689368010530057249524125613121
0692555165426329865997159318966021297187978698969150704428691492068511116 2
2040120833426489092427802185048384158498929909826610877614474089081595333
1250501916362625506881627797303648792319969938263851348283236880626111960 8
0651333799658356007922045766790422897142025201323366494910456776686229191
0991390326340735375731167075742451315359068421406707471987829194404184818
267822265625

Помним, что никогда не сравниваем вещественные числа, потому что у них может быть хвостик

>>> 1.1
1.1
>>> 2.2
2.2
>>> 1.1 + 2.2
3.3000000000000003
>>> 1.1 + 2.2 == 3.3
False

Следующий тип - строки. Вообще, с ними в питоне очень прикольно. Как такового символьного типа тут не существует. Вернее, он есть, но это однобайтовое число и вот эта фигня. С учётом того, что в питоне у нас утиная типизация (явно типы переменных не описываются, во всех переменных можно хранить любые объекты *потому что это вообще не хранение, а просто ссылки на объекты, которые можно перетыкать, куда вздумается*), у нас строки состоят из, та-да, строк. Соответственно, возникает прикол с обращением элемента по индексу - его можно делать, пока не надоест)))

```
>>> a = "qwert"
>>> a[4]
't'
>>> a[4][0]
't'
>>> a[4][0][0]
't'
>>> type(a[4][0][0])
<class 'str'>
```

При этом заметим, что у нас и одинарные, и двойные кавычки обозначают строку. Да, даже многобуквенную можно в одинарные записать, посмотри на последний вывод. При этом есть ещё многострочные кавычки, чтобы с энтерами ввод писать. Обозначаются они тройными парами - то есть *“””text”””* или *‘’’text’’’*

Строка - неизменяемый объект. Каждый раз у нас создаётся новый объект

```
>>> a = 'qwerty'
>>> id(a)
139821987357840
>>> a[0] = 'Q'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

```
>>> a = 'Hello'
>>> b = 'World'
>>> c = a + """, """ + b
>>> c
'Hello, World'
>>> type(c)
<class 'str'>
```

Дальше рассмотрим совокупности объектов. Неизменяемая совокупность объектов (слово “совокупность” тут не просто так - объекты могут быть разных типов) называется кортеж, обозначается набором значений в круглых скобках (также, по умолчанию, последовательность без скобок тоже считается кортежом). Изменяемая совокупность называется списком, обозначается квадратными скобками

```
>>> a = 1, 2, 3, "qwert"
>>> a
(1, 2, 3, 'qwert')
>>> type(a)
<class 'tuple'>

>>> b = [1, 2, 3, "qwe"]
>>> b
[1, 2, 3, 'qwe']
>>> type(b)
<class 'list'>

>>> b[2] = 4
>>> b
[1, 2, 4, 'qwe']

>>> a[2] = 4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Также у нас существует словарик - это объект формата ключ-значение, обозначается фигурными скобками

```
>>> d = {"qwe": 1, "False": "True"}
>>> d["False"]
'True'
>>> type(d)
<class 'dict'>
```

Поговорим про объекты как таковые. Питончик - ленивая штука, лишний раз ничего не создаёт. Поэтому неизменяемые объекты для каждой новой переменной он не создаёт по новой, а просто обоими указывает на этот неизменяемый объект. При этом для изменяемых, при этом равных объектов будут созданы свои собственные объекты.

Поиграемся сейчас со сравнением объектов. у нас для этого есть две операции: **==** сравнит объекты на равенство (“равность”, правильнее сказать), операция ***is*** сравнит id объекта, проверяя, это одно и то же или нет

```
>>> a = "qwer" # неизменяемая строка
>>> print(a)
qwer
>>> b = "qwer" # лень создавать новую - просто добавляется ссылка на область памяти с этой строкой
>>> type(a)
<class 'str'>
>>> type(b)
<class 'str'>

>>> a == b
True
>>> a is b
True

>>> a = [1, 2, 3, 4]
>>> b = [1, 2, 3, 4]
>>> type(a)
<class 'list'>
>>> type(b)
<class 'list'>

>>> a == b
True
>>> a is b
False
>>> # они равные, но не одно и то же
```

При этом присваивание вполне себе просто перевязывает ссылки на объекты, как неизменяемые, так и изменяемые

```
>>> a[0] = 123
>>> print(a, b)
[123, 2, 3, 4] [1, 2, 3, 4]

>>> a = b
>>> print(a, b)
[1, 2, 3, 4] [1, 2, 3, 4]
```

```
>>> a[0] = 123
>>> print(a, b)
[123, 2, 3, 4] [123, 2, 3, 4]

>>> a == b
True
>>> a is b
True

>>> b = [1, 2, 3, 4]
>>> a == b
False
>>> a is b
False
```

Немного про типы: то, что мы руками не задаём типы, не означает, что их нет. Просто все действия с ними в рамках работы переменных для нас скрыты, потому что переменная это просто имя для какой-то области памяти, где лежит объект. И тип у нас не у переменной, а у объекта. *Потому что погонял у настоящего мужика может быть много, а сам у себя он такой один*

При этом адекватные преобразования типов у нас есть

```
>>> s = "123"
>>> type(s)
<class 'str'>

>>> a = int(s)
>>> type(a)
<class 'int'>

>>> a * 2
246
```

Прикол: в питоне всё это объекты. Типы это тоже объекты)))

```
>>> typ = int
>>> typ is int # это одно и тоже теперь, один объект
True

>>> typ(s) # int это просто объект с функцией (), по которой он умеет преобразовывать типы. Раз теперь typ это такой же объект, у него есть такая же функция)))
123

>>> type(typ(s))
<class 'int'>
```

Функции

```
>>> def fun(a, b):
...     c = a * 2 + b
...     return c
>>> type(fun)
<class 'function'>

>>> fun
<function fun at 0x7ff9c8472dd0>

>>> fun(2, 3)
7
```

И да, функция это тоже объект)))))
Так что можно творить ту же фигню с присваиваниями

```
>>> fufufu = fun
>>> fufufu is fun
True
>>> fufufu(fun(2, 3), fufufu(3, 4))
24

>>> id(fun)
140710783692240
>>> id(fufufu)
140710783692240
```

Особенность функций Питончика: это просто алгоритм, а не типизированная структура. Поэтому суй, что хочешь, всё сработает. Утиная типизация, мать её)))

```
>>> fun("q", "w")
'qqw'

>>> fun("q", 2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in fun
TypeError: can only concatenate str (not "int") to str
can only concatenate str (not "int") to str
```

Он тут лёг не потому, что ошибка, а потому, чсто не умеет делать + для *int* и *str*
Типизация строгая, но динамическая. Типизации имён в Питоне нет

Связывание объектов

>>> 100 +(1 + 2) * 3 # сколько объектов создалось и удалилось между этой строчкой...
109
>>> # ... и этой?

Как сохраняется объект? Пока на него есть ссылки,операнд **=** делает связывание имени с объектом

>>> a = 109
>>> b = a
>>> b is a
True

>>> del a # удаляет ИМЯ, не объект
>>> b
109

>>> del b
>>> # вот теперь объекта 109 не существует

>>> # посмотреть, какие имена мы помним нашим неймспейсом - dir()
>>> dir()
['_', '_12', '_13', '_14', '_15', '_17', '_18', '_2', '_22', '_23', '_26', '_29', '_30', '_33', '_35', '_39', '_4', '_40', '_43', '_44', '_45', '_46', '_50', '_51', '_52', '_53', '_55', '_56', '_6', '_60', '_62', '_63', '_66', '_67', '_68', '_7', '_71', '_72', '_73', '_75', '_76', '_77', '_78', '_8', '_80', '_83', '_85', '_89', '_9', '_91', '__annotations__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__requires__', '__spec__', 'd', 'distribution', 'fufufu', 'fun', 'get_ptpython', 'importlib_load_entry_point', 'load_entry_point', 're', 's', 'sys', 'typ']

>>> a = 13
>>> dir(a)
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__', '__delattr__', '__dir__', '__divmod__', '__doc__', '__eq__', '__float__', '__floor__', '__floordiv__', '__format__', '__ge__', '__getattribute__', '__getnewargs__', '__gt__', '__hash__', '__index__', '__init__', '__init_subclass__', '__int__', '__invert__', '__le__', '__lshift__', '__lt__', '__mod__', '__mul__', '__ne__', '__neg__', '__new__', '__or__', '__pos__', '__pow__', '__radd__', '__rand__', '__rdivmod__', '__reduce__', '__reduce_ex__', '__repr__', '__rfloordiv__', '__rlshift__', '__rmod__', '__rmul__', '__ror__', '__round__', '__rpow__', '__rrshift__', '__rshift__', '__rsub__', '__rtruediv__', '__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__truediv__', '__trunc__', '__xor__', 'as_integer_ratio', 'bit_count', 'bit_length', 'conjugate', 'denominator', 'from_bytes', 'imag', 'numerator', 'real', 'to_bytes']

>>> # посчитаем кол-во ссылок

>>> a = b = c = "qwe"
>>> import sys
>>> sys.getrefcount(a)
4

>>> del b
>>> sys.getrefcount(a)
3

>>> # 4 и 3, тк getrafcount сам имеет ссылку на объект, чтобы работать

Множественное связывание - может сразу многим переменным присваивать много значений. Обязательно делать это имя в имя, количества с обеих сторон одинаковы

>>> a, b, c = 1, 2, 3
>>> a
1
>>> b
2
>>> c
3

>>> def funn(a, b):
...   return a, b, a, b
>>> a, b, c, d = funn(1, 2)
>>> a
1
>>> b
2
>>> c
1
>>> d
2

>>> a, b, c = funn(1, 2)
Traceback (most recent call last):
*** File "<stdin>", line 1, in <module>***
ValueError: too many values to unpack (expected 3)

too many values to unpack (expected 3)

хак по количеству параметров - делать специальные переменные, которые будут сохранять списки параметров

```
>>> a, *b, c = funn(1, 2)
>>> a
1
>>> c
2
>>> b
[2, 1]

>>> a = b = [a, 2, 3, "qwe"]
>>> a
[1, 2, 3, 'qwe']
>>> b
[1, 2, 3, 'qwe']
>>> a is b
True

>>> c = b
>>> c is b
True


>>> a
[1, 2, 3, 'qwe']
>>> b
[1, 2, 3, 'qwe']
>>> c
[1, 2, 3, 'qwe']

>>> a[2] = 100500
>>> a
[1, 2, 100500, 'qwe']
>>> b
[1, 2, 100500, 'qwe']
>>> c
[1, 2, 100500, 'qwe']
```

Ещё один прикол - рекурсивные ссылки

```
>>> l = [1, 2]
>>> m = [l, 3]
>>> l[0] = m
>>> l
[[[...], 3], 2]

>>> m
[[[...], 2], 3]
```