```
# --Loops--
# По същество циклите са начин да правите нещо отново и отново.
# Започнете с въвеждане на код cat.py в прозореца на терминала.
# В текстовия редактор започнете със следния код:
"""
print("meow")
print("meow")
print("meow")

"""
# Изпълнявайки този код, като напишете python cat.py, ще забележите, че програмата изпълнява meow три пъти.
# Когато се развивате като програмист, вие искате да обмислите как човек може да подобри области от своя код,
# където въвежда едно и също нещо отново и отново.
# Представете си къде някой би искал да „meow“ 500 пъти.
# Би ли било логично да въвеждате същия израз на print("meow") отново и отново?
# Циклите ви позволяват да създадете блок от код, който се изпълнява отново и отново.

# --While Loops--
# Цикълът while е почти универсален във всички кодиращи езици.
# Такъв цикъл ще повтаря блок от код отново и отново.
# В прозореца на текстовия редактор редактирайте кода си, както следва:
"""
i = 3
while i != 0:
    print("meow")
"""
# Забележете как въпреки че този код ще изпълни print("meow") няколко пъти, той никога няма да спре!
# Ще се върти завинаги. докато циклите работят, като многократно питат дали условието на цикъла е изпълнено.
# В този случай компилаторът пита „не е ли равно на нула?“
# Когато заседнете в цикъл, който се изпълнява вечно, можете да натиснете control-c на клавиатурата си, за да излезете от цикъла.
# За да коригираме този цикъл, който продължава вечно, можем да редактираме нашия код, както следва
"""
i = 3
while i != 0:
    print("meow")
    i = i - 1
"""
# Забележете, че сега нашият код се изпълнява правилно, намалявайки i с 1 за всяка „итерация“ през цикъла.
# Този термин итерация има специално значение в рамките на кодирането.
# Под итерация имаме предвид един цикъл през цикъла.
# Първата итерация е "0-та" итерация през цикъла.
# Втората е „1-вата“ итерация.
# В програмирането ние броим, започвайки с 0, след това 1, след това 2
```

```
43 .
44 # Можем допълнително да подобрим нашия код, както следва:
45 """"""""""
46 i = 1
47     while i <= 3:
48         print("meow")
49         i = i + 1
50 """"""""""
51 # Забележете, че когато кодираме i = i + 1, присвояваме стойността на
   i отдясно наляво.
52 # По-горе започваме от едно, както повечето хора броят (1, 2, 3).
53 # Ако изпълните кода по-горе, ще го видите да мяука три пъти.
54 # Най-добрата практика в програмирането е да започнете да броите с
   нула.
55 # Можем да подобрим нашия код, за да започнем да броим с нула:
56 """"""""""
57 i = 0
58 while i < 3:
59     print("meow")
60     i += 1
61 """"""""""
62 # Забележете как промяната на оператора на i < 3 позволява на нашия
   код да функционира по предназначение.
63 # Започваме с броене с 0 и той итерира нашия цикъл три пъти,
   произвеждайки три мяукания.
64 # Освен това забележете как i += 1 е същото като i = i + 1.
65
66 # -- For Loops --
67 # Цикълът for е различен тип цикъл.
68 # За да разберете най-добре for цикъла, най-добре е да започнете, като
    говорите за нов тип променлива, наречен списък в Python.
69 # Както и в други области на нашия живот, можем да имаме списък с
   хранителни стоки, списък със задачи и т.н.
70 # Цикълът for преминава през списък от елементи.
71 # Например, в прозореца на текстовия редактор, променете вашия cat.py
   код, както следва:
72 """"""""""
73 for i in [0, 1, 2]:
74     print("meow"
75 """"""""""
76 # Забележете колко изчистен е този код в сравнение с предишния код на
   цикъла while.
77 # В този код i започва с 0, meow, на i се присвоява 1, meow и накрая
   на i се присвоява 2, meow и след това завършва.
78 # Докато този код постига това, което искаме, има някои възможности за
    подобряване на нашия код за екстремни случаи.
79 # На пръв поглед нашият код изглежда страхотно. Но какво ще стане, ако
    искате да повторите до един милион?
80 # Най-добре е да създадете код, който може да работи с такива
   екстремни случаи.
81 # Съответно можем да подобрим нашия код, както следва:
82 """"""""""
83 for i in range(3):
```

```
    print("meow")
"""
# Забележете как range(3) автоматично връща три стойности (0, 1 и 2).
# Този код ще се изпълни и ще произведе желания ефект, мяукайки три пъти.
# Нашият код може да бъде допълнително подобрен. Забележете как никога не използваме i изрично в нашия код.
# Тоест, докато Python се нуждае от i като място за съхраняване на номера на итерацията на цикъла,
# ние никога не го използваме за други цели.

# В Python, ако такава променлива няма никакво друго значение в нашия код,
# можем просто да представим тази променлива като единична долна черта _.
# Следователно можете да промените кода си, както следва:
"""
for _ in range(3):
    print("meow")
"""
# Забележете как промяната на i на _ няма никакво влияние върху функционирането на нашата програма.

# Нашият код може да бъде допълнително подобрен. За да разширите ума си към възможностите в Python,
# разгледайте следния код:
"""
print("meow" * 3)
"""
# Забележете как ще мяука три пъти, но програмата ще произведе мяумяумяу като резултат.
# Помислете: Как можете да създадете прекъсване на ред в края на всяко мяукане?

# Всъщност можете да редактирате кода си, както следва:
"""
print("meow\n" * 3, end="")
"""
# Забележете как този код произвежда три мяукания, всяко на отделен ред.
# Чрез добавяне на end="" и \n казваме на компилатора да добави нов ред в края на всяко мяукане.

# Подобряване с въвеждане от потребителя

# Може би искаме да получим информация от нашия потребител.
# Можем да използваме цикли като начин за валидиране на входа на потребителя.
# Обща парадигма в рамките на Python е използването на цикъл while за валидиране на въведеното от потребителя.
# Например, нека опитаме да подканим потребителя за число, по-голямо или равно на 0:
"""
```

```
while True:
    n = int(input("What's n? "))
    if n < 0:
        continue
    else:
        break
"""""""""
# Забележете, че въведохме две нови ключови думи в Python, continue и break.
# continue изрично казва на Python да премине към следващата итерация на цикъл.
# break, от друга страна, казва на Python да "излезе" от цикъл рано, преди да е завършил всичките си итерации.
# В този случай ще продължим към следващата итерация на цикъла, когато n е по-малко от 0 - в крайна сметка
# ще подканим отново потребителя с „What's n?“.
# Ако обаче n е по-голямо или равно на 0, ще излезем от цикъла
# и ще позволим на останалата част от нашата програма да работи.

# Оказва се, че ключовата дума continue е излишна в този случай. Можем да подобрим нашия код, както следва:
"""""""""
while True:
    n = int(input("What's n? "))
    if n > 0:
        break

for _ in range(n):
    print("meow")
"""""""""
# Забележете как този цикъл while ще работи винаги (завинаги), докато n е по-голямо от 0.
# Когато n е по-голямо от 0, цикълът прекъсва.

# Внасяйки нашето предишно обучение, можем да използваме функции за допълнително подобряване на нашия код:
"""""""""
def main():
    meow(get_number())

def get_number():
    while True:
        n = int(input("What's n? "))
        if n > 1:
            return n

def meow(n):
    for _ in range(n):
        print("meow")

main()
"""""""""
# Забележете как не само променихме вашия код, за да работи в
```

```
168 множество функции,
169 # но също така използвахме оператор return, за да върнем стойността
    на n обратно към основната функция.
170
171
172 # -- Повече за списъците - More About Lists --
173
174 # Помислете за света на Хогуортс от прочутата вселена на Хари Потър.
175 # В терминала въведете код hogwarts.py. В текстовия редактор
    кодирайте както следва:
176 """"""""""
177 students = ["Hermoine", "Harry", "Ron"]
178
179 print(students[0])
180 print(students[1])
181 print(students[2])
182
183 """"""""""
184 # Забележете как имаме списък със студенти с техните имена, както по-
    горе.
185 # След това отпечатваме ученика, който е на 0-то място, „Hermoine“.
    Всеки от другите студенти също е отпечатан.
186 #
187 # Точно както илюстрирахме по-рано, можем да използваме цикъл, за да
    обикаляме списъка.
188 # Можете да подобрите кода си, както следва:
189 """"""""""
190 students = ["Hermoine", "Harry", "Ron"]
191
192 for student in students:
193     print(student)
194 """"""""""
195 # Забележете, че за всеки ученик в списъка със студенти, той ще
    отпечата ученика по предназначение.
196 # Може би се чудите защо не използвахме обозначението _, както беше
    обсъдено по-горе.
197 # Избираме да не правим това, защото student се използва изрично в
    нашия код.
198
199 # Можете да научите повече в документацията на Python за списъци.
200 # https://docs.python.org/3/tutorial/datastructures.html#more-on-
    lists
201
202 # -- Length -- Дължина --
203
204 # Можем да използваме len като начин за проверка на дължината на
    списъка, наречен студенти.
205 # Представете си, че не просто искате да отпечатате името на ученика
    , но и неговата позиция в списъка.
206 # За да постигнете това, можете да редактирате кода си, както следва:
207 """"""""""
208 students = ["Hermoine", "Harry", "Ron"]
209
```

```
for i in range(len(students)):
    print(i + 1, students[i])
"""
# Забележете как изпълнението на този код води не само до получаване на позицията на всеки ученик плюс едно чрез i + 1,
# но също така отпечатва името на всеки ученик.
# len ви позволява динамично да виждате колко дълъг е списъкът на учениците, независимо колко расте.

# Можете да научите повече в документацията на Python за len.
# https://docs.python.org/3/library/functions.html?highlight=len#len


# -- Dictionaries -- РЕЧНИЦИ --

# dicts или Dictionaries е структура от данни, която ви позволява да свързвате ключове със стойности.
# Когато списъкът е списък с множество стойности, dict свързва ключ със стойност.
# Имайки предвид къщите на Хогуортс, можем да назначим конкретни ученици в конкретни къщи.
"""
students = ["Hermoine", "Harry", "Ron", "Draco"]
houses = ["Gryffindor", "Gryffindor", "Griffindor", "Slytherin"]
"""
# Забележете, че можем да обещаем, че винаги ще поддържаме тези списъци в ред.
# Индивидът на първата позиция от учениците се свързва с къщата на първата позиция в списъка с къщи и т.н.
# Това обаче може да стане доста тромаво, тъй като списъците ни растат!
#
# Можем да подобрим нашия код с помощта на dict, както следва:
"""
students = {
    "Hermoine": "Gryffindor",
    "Harry": "Gryffindor",
    "Ron": "Gryffindor",
    "Draco": "Slytherin",
}
print(students["Hermoine"])
print(students["Harry"])
print(students["Ron"])
print(students["Draco"])

output:
Gryffindor
Gryffindor
Gryffindor
Slytherin
"""

# Забележете как използваме {} фигурни скоби, за да създадем речник.
```

```
255 # Когато списъците използват числа, за да обикалят списъка, dicts ни
    позволяват да използваме думи.
256 # Стартирайте кода си и се уверете, че резултатът ви е както следва:
257
258 # Можем да подобрим нашия код, както следва:
259 """"""""""
260 students = {
261     "Hermoine": "Gryffindor",
262     "Harry": "Gryffindor",
263     "Ron": "Gryffindor",
264     "Draco": "Slytherin",
265 }
266 for student in students:
267     print(student)
268
269 output:
270 Hermoine
271 Harry
272 Ron
273 Draco
274 """"""""""
275 # Забележете как изпълнявайки този код, for цикълът ще обхожда само
    всички ключове,
276 # което ще доведе до списък с имената на учениците. Как можем да
    отпечатаме и стойности, и ключове?
277
278 # Променете кода си, както следва:
279 """"""""""
280 students = {
281     "Hermoine": "Gryffindor",
282     "Harry": "Gryffindor",
283     "Ron": "Gryffindor",
284     "Draco": "Slytherin",
285 }
286 for student in students:
287     print(student, students[student])
288
289 output:
290 Hermoine Gryffindor
291 Harry Gryffindor
292 Ron Gryffindor
293 Draco Slytherin
294 """"""""""
295 # Забележете как ученици students[student] ще отидат до ключа на
    всеки ученик и ще намерят стойността на къщата им.
296 # Изпълнете кода си и ще забележите как изходът е малко объркан.
297 # Можем да изчистим функцията за печат, като подобрим нашия код,
    както следва:
298 """"""""""
299 students = {
300     "Hermoine": "Gryffindor",
301     "Harry": "Gryffindor",
302     "Ron": "Gryffindor",
```

```
303     "Draco": "Slytherin",
304 }
305 for student in students:
306     print(student, students[student], sep=", ")
307
308 output:
309 Hermoine, Gryffindor
310 Harry, Gryffindor
311 Ron, Gryffindor
312 Draco, Slytherin
313 """""""""
314
315 # Забележете как това създава чисто разделяне на , между всеки
    отпечатан елемент.
316
317 # Ами ако имаме повече информация за нашите ученици? Как бихме могли
    да свържем повече данни с всеки от учениците?
318 """""""""
319     | name      | house         | patronus
320 0   |Harmione   |   Gryffindor  |   Otter
321 1   |Harry      |   Gryffindor  |   Stag
322 2   |Ron        |   Gryffindor  |   Jack Russell terrier
323 3   |Draco      |   Slytherin   |
324
325 """""
326 # Можете да си представите, че искате да имате много данни, свързани
    с множество неща с един ключ.
327 # Подобрете кода си, както следва:
328 """""
329 students = [
330     {"name": "Hermoine", "house": "Gryffindor", "patronus": "Otter"},
331     {"name": "Harry", "house": "Gryffindor", "patronus": "Stag"},
332     {"name": "Ron", "house": "Gryffindor", "patronus": "Jack Russell
    terrier"},
333     {"name": "Draco", "house": "Slytherin", "patronus": None},
334 ]
335 """""
336 # Забележете как този код създава list  от dicts. Списъкът, наречен
    студенти, има четири dicts  в него:
337 # По един за всеки ученик. Също така забележете, че Python има
    специално обозначение None,
338 # където няма стойност, свързана с ключ.
339 #
340 # Сега имате достъп до множество интересни данни за тези ученици.
    Сега допълнително променете кода си, както следва:
341 """""
342 students = [
343     {"name": "Hermoine", "house": "Gryffindor", "patronus": "Otter"},
344     {"name": "Harry", "house": "Gryffindor", "patronus": "Stag"},
345     {"name": "Ron", "house": "Gryffindor", "patronus": "Jack Russell
    terrier"},
346     {"name": "Draco", "house": "Slytherin", "patronus": None},
347 ]
```

```
348
349 for student in students:
350     print(student["name"], student["house"], student["patronus"], sep
    =", ")
351 """"""
352 # Забележете как цикълът for ще премине през всеки от dicts в списъка
    , наречен студенти.
353 # Можете да научите повече в документацията на Python за dicts.
354 # https://docs.python.org/3/tutorial/datastructures.html#dictionaries
355
356
357
```