Kontrolowanie akcji

Do tej pory dużo mówiliśmy o przechowywaniu informacji w komputerach, głównie w zmiennych, z którymi może pracować Python i komputer. Posiadanie informacji w formie, z którą komputer może pracować, jest z pewnością kluczowe dla nakłonienia komputera do zrobienia czegokolwiek. Pomyśl o tym jako o części „posiadanie" – posiadanie pewnych informacji, z którymi można pracować. Ale teraz musimy zwrócić uwagę na część „działania" . . . faktycznie pracując z tymi informacjami, aby stworzyć coś użytecznego lub rozrywkowego. Omówimy najważniejsze i najczęściej używane operacje, które pozwalają komputerowi robić różne rzeczy. Zaczynamy od czegoś, co komputery robią dobrze, robią szybko i robią dużo – podejmują decyzje.

Główne operatory kontrolujące działanie

Kontrolujesz, co robi twój program (i komputer), podejmując decyzje, co często wiąże się z dokonywaniem porównań. Do porównań używamy operatorów, takich jak te w Tabeli. Są one często określane jako operatory relacyjne lub operatory porównania, ponieważ porównując elementy, komputer określa jak dwie pozycje są powiązane.

Operator : Znaczenie

== : jest równe

!= : nie jest równe

< : jest mniejsze niż

> : jest większe niż

<= : jest mniejsze lub równe

>= : jest większe lub równe

Python oferuje również trzy operatory logiczne, zwane także operatorami boolowskimi, które umożliwiają ocenę wielokrotnych porównań przed podjęciem ostatecznej decyzji. Operatory te używają angielskiego słowa, no cóż, w zasadzie to, co mają na myśli, jak pokazano w tabeli

Operator : Znaczenie

and : oba są prawdziwe

or : jedno lub drugie jest prawdziwe

not : nie jest prawdą

Jeśli zastanawiasz się nad słowem Boolean, jest to odniesienie do faceta o imieniu George Boole, który w połowie XIX wieku pomógł ustanowić algebrę logiki, która w dużej mierze położyła podwaliny pod dzisiejsze komputery. Wszystkie te operatory są często używane w połączeniu z decyzjami if ... then ... else , aby dokładnie kontrolować, co robi aplikacja lub program. Aby podejmować takie decyzje, użyj instrukcji if w Pythonie

Podejmowanie decyzji z if

Słowo if jest często używane we wszystkich aplikacjach i programach komputerowych do podejmowania decyzji. Najprostsza składnia if to:

if warunek: zrób to

rób to bez względu na wszystko

Tak więc pierwsza linia do tej linii jest wykonywana tylko wtedy, gdy warunek jest spełniony. Jeśli warunek jest fałszywy, pierwsze działanie jest ignorowane. Niezależnie od tego, jaki okaże się warunek, w następnej kolejności wykonywana jest druga linia. Zauważ, że żaden wiersz nie jest wcięty. To wiele znaczy w Pythonie, jak wkrótce zobaczysz. Ale najpierw zróbmy kilka prostych przykładów z tą prostą składnią. Możesz spróbować sam w notatniku Jupyter lub pliku .py, jeśli chcesz kontynuować. Poniższy rysunek przedstawia prosty przykład, w którym zmienna o nazwie sun otrzymuje ciąg „down". Następnie instrukcja if sprawdza, czy zmienna sun zawiera słowo down, a jeśli tak, wypisuje "Good night!" Potem po prostu kontynuuje normalne drukowanie. Jestem tutaj. W wyniku widać, że wyświetlane są obie linie.

Upewnij się, że zawsze używasz dwóch znaków równości bez spacji między (==) do testowania równości. Łatwo o tym zapomnieć. Jeśli wpiszesz to źle, nie będzie działać zgodnie z oczekiwaniami. Jeśli uruchomisz ten sam kod ze słowem innym niż down w zmiennej sun, to pierwszy wydruk zostanie zignorowany, ale następny wiersz jest nadal wykonywany normalnie, ponieważ nie jest zależny od spełnienia warunku, jak pokazano na rysunku

W drugim przykładzie nie jest prawdą, że zmienna o nazwie sun zawiera True, dlatego reszta tej linii jest ignorowana i wykonywana jest tylko następna linia. Ta składnia, w której kod, który ma zostać wykonany, gdy warunek okaże się spełniony, znajduje się w tym samym wierszu co if, ale często chcesz zrobić więcej niż jedną rzecz, gdy warunek okaże się spełniony. W tym celu musisz wciąć każdą linię, aby została wykonana tylko wtedy, gdy warunek okaże się spełniony. I kod, który nie ma wcięcia poniżej if, jest wykonywany bez względu na to, czy warunek okaże się spełniony, czy nie. Zaleca się wcięcie o cztery spacje, ale nie jest to trudna i szybka zasada. Trzeba tylko pamiętać, że każda linia musi być wcięta tak samo. Możesz również użyć składni „wciętej", nawet jeśli tylko jeden wiersz kodu ma być wykonany, jeśli warunek okaże się spełniony. W rzeczywistości jest to najczęstszy sposób pisania if w Pythonie, ponieważ większość ludzi zgadza się, że czyni kod bardziej „czytelnym" z ludzkiego punktu widzenia. Tak naprawdę składnia to

if warunek:

zrób to

...

rób to bez względu na wszystko

Więc jeśli warunek okaże się spełniony, ten wiersz jest wykonywany, tak jak wszystkie inne wiersze, które mają równe wcięcie. Pierwsza linia bez wcięcia pod if jest wykonywana bez względu na wszystko. Możesz więc napisać prosty przykład słońca w ten sposób:

sun = "down"

if sun == "down":

print("Good night!")

print("I am here")

Jak widać na rysunku , kod działa dokładnie tak samo, jak umieszczenie kodu w jednej linii. Jeśli słońce zachodzi, to dobranoc! drukuje przed wykonaniem drugiego wydruku. Jeśli słońce nie zatrzymuje się w dół, to drukiem na Dobranoc! jest pomijany i ignorowany.

Jeśli zastanawiasz się, która metoda jest lepsza, zależy to od tego, co rozumiesz przez lepsze. Jeśli masz na myśli lepsze, jeśli chodzi o to, która metoda działa najszybciej, to nie. Podczas wykonywania kodu nie zobaczysz żadnej różnicy prędkości. Jeśli przez lepsze masz na myśli „łatwiejszy do czytania dla ludzkiego programisty". wtedy większość ludzi prawdopodobnie skłania się ku drugiej metodzie z kodem wciętym pod instrukcją if. Pamiętaj, że możesz wciąć dowolną liczbę wierszy pod if, a te wcięte wiersze są wykonywane tylko wtedy, gdy warunek okaże się spełniony. Jeśli warunek okaże się fałszem, żaden z wciętych wierszy nie zostanie wykonany. Kod pod wciętymi liniami jest zawsze wykonywany, ponieważ nie jest zależny od warunku. Oto przykład, w którym mamy cztery wiersze kodu, które są wykonywane tylko wtedy, gdy warunek okaże się spełniony:

total = 100

sales_tax_rate = 0.065

taxable = True

if taxable:

print(f"Subtotal : ${total:.2f}")

sales_tax = total * sales_tax_rate

print(f"Sales Tax: ${sales_tax:.2f}")

total = total sales_tax

print(f"Total : ${total:.2f}")

Musisz przeliterować True i Fals z początkowej wielkiej litery, a resztę małymi literami. Jeśli wpiszesz go w inny sposób, Python nie rozpozna go jako Boolean True lub False, a Twój kod nie będzie działał zgodnie z oczekiwaniami. Zauważ, że w instrukcji if użyliśmy

if taxable

Jest to całkowicie w porządku, ponieważ podaliśmy opodatkowaniu wartość logiczną, która może być tylko prawdą lub fałszem. Możesz zobaczyć, jak inni wpisują to jako

if taxable == True:

To też jest w porządku i nie będzie miało żadnego negatywnego wpływu na kod. Wartość == True jest po prostu niepotrzebna, ponieważ sama w sobie opodatkowanie jest już prawdą lub nie fałszem. W każdym razie, jak widać, zaczynamy od total, sales_tax_rate i zmiennej podlegającej opodatkowaniu. Gdy opodatkowanie to prawda, wszystkie cztery wiersze pod if są wykonywane i otrzymujesz wynik pokazany na rysunku

```
total = 100
sales_tax_rate = 0.065
taxable = True
if taxable:
    print(f"Subtotal : ${total:.2f}")
    sales_tax = total * sales_tax_rate
    print(f"Sales Tax: ${sales_tax:.2f}")
    total = total + sales_tax
print(f"Total    : ${total:.2f}")
```

```
Subtotal : $100.00
Sales Tax: $6.50
Total    : $106.50
```

Gdy opodatkowanie jest ustawione na False, wszystkie wcięte wiersze są pomijane, a pokazana suma jest oryginalną sumą bez żadnego podatku od sprzedaży, jak pokazano na rysunku

```
total = 100
sales_tax_rate = 0.065
taxable = False
if taxable:
    print(f"Subtotal : ${total:.2f}")
    sales_tax = total * sales_tax_rate
    print(f"Sales Tax: ${sales_tax:.2f}")
    total = total + sales_tax
print(f"Total    : ${total:.2f}")
```

```
Total    : $100.00
```

Nawiasy klamrowe i elementy .2f na rysunkach wcześniejszych służą tylko do formatowania, i nie mają nic wspólnego z logiką if kodu.

Dodawanie else do Twojego loginu if

Do tej pory przyjrzeliśmy się przykładom kodu, w których kod jest wykonywany, jeśli jakiś warunek okaże się spełniony. Jeśli warunek okaże się fałszem, kod zostanie zignorowany. Czasami możesz mieć

sytuację, w której chcesz, aby jeden fragment kodu został wykonany, jeśli warunek okaże się spełniony, w przeciwnym razie (else), jeśli nie okaże się prawdziwy, chcesz wykonać inny fragment kodu. W takim przypadku możesz dodać else: do swojego if. Wszelkie wiersze kodu bez wcięcia pod else: są wykonywane tylko wtedy, gdy warunek nie okazał się spełniony. Oto logika i składnia:

if warunek:

zrób tutaj wcięte linie

...

else:

zrób tutaj wcięte linie

...

zrób pozostałe linie bez wcięć bez względu na wszystko.

Rysunek poniżej pokazuje prosty przykład, w którym pobieramy aktualny czas z zegara komputera za pomocą datetime.now(). Jeśli godzina tego czasu jest mniejsza niż 12, program pokazuje Good morning! W przeciwnym razie pokazuje Good afternoon! Niezależnie od godziny drukuje mam nadzieję, że dobrze sobie radzisz. Więc jeśli napiszesz taki program i uruchomisz go rano, otrzymasz odpowiednie powitanie, a następnie mam nadzieję, że dobrze sobie radzisz, jak na rysunku

```
import datetime as dt
# Get the current date and time
now = dt.datetime.now()
# Make a decision based on hour
if now.hour < 12:
    print("Good morning")
else:
    print("Good afternoon")
print("I hope you are doing well!")
```

```
Good morning
I hope you are doing well!
```

Teraz możesz na to spojrzeć i powiedzieć „Wow, to naprawdę imponujące, Einstein. Ale co, jeśli w nocy jest 11:00? Czy naprawdę chcesz powiedzieć „Dzień dobry”? Jeszcze jedno pytanie zasługujące na głośne Hmm. To, czego potrzebujemy, to if. . . else gdzie indziej istnieje wiele innych możliwości. W tym miejscu w grę wchodzi stwierdzenie elif.

Obsługa wielu innych else z elif

Kiedy if . . . else nie wystarczy, aby obsłużyć wszystkie możliwości, istnieje elif (które, jak można się domyślić, jest słowem utworzonym z else if. Instrukcja if może zawierać dowolną liczbę warunków elif. Możesz dołączyć lub nie dołączyć końcowego Instrukcja else, która jest wykonywana tylko wtedy, gdy if i wszystkie poprzednie elif okażą się fałszywe.W najprostszej postaci składnia dla if z elif i else jest

if warunek:

zrób te wcięte linie kodu

...

warunek elif

zrób te wcięte linie kodu

...

zrób te wiersze kodu bez wcięć bez względu na wszystko.

Biorąc pod uwagę tę strukturę, możliwe jest, że żaden z wciętych kodów nie zostanie wykonany. Spójrz na ten przykład:

```
light_color = "green"

if light_color == "green":

print("Go")

elif light_color == "red":

print("Stop")

print("This code executes no matter what")
```

Wykonanie tego kodu skutkuje:

```
Go

This code executes no matter what
```

Jeśli zmienisz kolor światła na czerwony, w ten sposób:

```
light_color = "red"

if light_color == "green":

print("Go")

elif light_color == "red":

print("Stop")

print("This code executes no matter what")
```

. . . to wynik jest

```
Stop

This code executes no matter what
```

Załóżmy, że zmieniasz kolor światła na inny niż czerwony lub zielony w następujący sposób:

```
light_color = "yellow"

if light_color == "green":

print("Go")

elif light_color == "red":

print("Stop")
```

```
print("This code executes no matter what")
```

Wykonanie tego kodu daje następujący wynik, ponieważ ani color==„green” ani color==„red” nie okazały się prawdziwe, więc żaden z wciętych kodów nie został wykonany:

```
This code executes no matter what
```

Możesz dodać opcję else, która ma miejsce tylko wtedy, gdy wszystkie poprzednie warunki okażą się fałszywe, na przykład:

```
light_color = "yellow"

if light_color == "green":

print("Go")

elif light_color == "red":

print("Stop")

else:

print("Proceed with caution")

print("This code executes no matter what")
```

Dane wyjściowe to:

```
Proceed with caution

This code executes no matter what
```

Fakt, że light_color jest żółty, uniemożliwia spełnienie dwóch pierwszych warunków if, więc wykonywany jest tylko kod else. Dotyczy to wszystkiego, co umieścisz w zmiennej light_color, ponieważ else nie szuka określonego warunku. To po prostu odgrywanie roli w logice „jeśli wszystko inne zawiedzie, zrób to”.

Operacje trójskładnikowe

Tu nie zakładamy, że znasz inne języki programowania, ale musimy wspomnieć o tym czytelnikom, którzy znają inne języki. Wiele języków ma skrócony sposób robienia if. . . else wszystko w jednym wierszu kodu. Rozważmy na przykład następujący kod JavaScript:

```
//Przykład kodu JavaScript, nie działa w Pythonie

age=14;

beverage = (age > 20) ? "beer" : "milk";

alert("Have a " beverage);
```

Trzecia linia to skrócony sposób powiedzenia „Włóż do napoju zmienną piwo lub mleko w zależności od zawartości zmiennej wiek” W Pythonie możesz napisać to jako mniej więcej tak:

```
age = 31

if age < 21:

beverage = "milk"
```

```
elif age >= 21 and age < 80:

beverage = "beer"

else:

beverage = "prune juice"

print("Have a " beverage)
```

Kod jest dłuższy i bardziej rozwlekły, ale łatwiejszy do zrozumienia. Dodawanie komentarzy, które zawsze jest opcją, również bardzo pomaga, w następujący sposób:

```
age = 31

if age < 21:

# If under 21, no alcohol

beverage = "milk"

elif age >= 21 and age < 80:

# Ages 21 - 79, suggest beer

beverage = "beer"

else:

# If 80 or older, prune juice might be a good choice.

beverage = "prune juice"

print("Have a " beverage)
```

Jeśli zastanawiasz się, jaka jest zasada wcinania komentarzy, nie ma reguły. Komentarze to tylko notatki do ciebie, nie są kodem wykonywalnym. Dlatego nigdy nie są wykonywane jak kod, bez względu na wcięcie na poziomie.

Powtarzanie procesu z for

Podejmowanie decyzji to duża część pisania wszelkiego rodzaju aplikacji, czy to gier, sztucznej inteligencji, robotyki . . . cokolwiek. Ale są też przypadki, w których trzeba liczyć lub wykonywać jakieś zadanie w kółko. Pętla for jest jednym ze sposobów na zrobienie tego. Pozwala powtórzyć linię kodu lub kilka linii kodu tyle razy, ile lubisz.

Zapętlanie liczb w range

Jeśli wiesz, ile razy chcesz, aby pętla się powtarzała, użycie tej składni może być najłatwiejsze:

```
for x in range(y):

Zrób to

Zrób to

...

kod bez wcięcia jest wykonywany po pętli
```

Zastąp x dowolną wybraną nazwą zmiennej. Zastąp y dowolną liczbą lub zakresem liczb. Jeśli określisz jedną liczbę, zakres będzie wynosił od zera do jednego mniej niż ostateczna liczba. Na przykład uruchom ten kod w notatniku Jupyter lub pliku .py:

```
for x in range(7):

print(x)

print("All done")
```

Wynik jest wynikiem wykonania print(x) raz dla każdego przejścia przez pętlę, gdzie x zaczyna się od zera. Ostatnia linia, która nie jest wcięta, jest wykonywana po zakończeniu pętli. Wynik to:

```
0

1

2

3

4

5

6

Wszystko gotowe
```

Jeśli uznasz to za trochę irytujące, ponieważ pętla nie robi tego, co chciałeś, możesz umieścić dwie liczby oddzielone przecinkiem jako zakres. Pierwsza liczba to miejsce, w którym zaczyna się liczenie pętli. Druga liczba jest o jeden większa niż miejsce, w którym pętla się zatrzymuje (co jest niefortunne dla czytelności, ale takie jest życie). Na przykład oto pętla for z dwiema liczbami w zakresie:

```
for x in range(1, 10):

print(x)

print("All done")
```

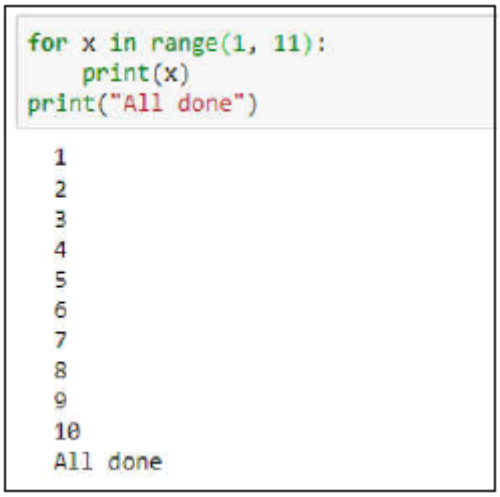
Po uruchomieniu tego kodu licznik zaczyna się od 1 i, jak wskazano, zatrzymuje się przed ostatnią liczbą:

```
1

2

3

4

5

6

7

8

9
```

All done

Jeśli naprawdę chcesz, aby pętla liczyła od 1 do 10, zakres będzie musiał wynosić 1,11.

To nie sprawi, że twoje komórki mózgowe będą szczęśliwsze, ale przynajmniej osiągnie pożądany cel od 1 do 10, jak pokazano na rysunku



Pętla przez string

Użycie range() w pętli for jest opcjonalne. Zakres można zastąpić ciągiem, a pętla powtarza się raz dla każdego znaku w ciągu. Zmienna x (lub jakkolwiek nazwiesz zmienną) zawiera jeden znak z ciągu przy każdym przejściu przez pętlę, od lewej do prawej. Składnia tutaj to:

for x w ciągu znaków

Zrób to

Zrób to

...

Zrób to po zakończeniu pętli

Jak zwykle zastąp x dowolną nazwą zmiennej. Ciąg powinien być tekstem ujętym w cudzysłów lub powinien być nazwą zmiennej zawierającej ciąg. Na przykład wpisz ten kod do notesu Jupyter lub pliku .py:

for x w "snorkel":

print(x)

print("Done")

Po uruchomieniu tego kodu otrzymasz następujące dane wyjściowe. Pętla drukowała jedną literę ze słowa „snorkel” przy każdym przejściu przez pętlę. Po zakończeniu zapętlania wykonanie spadło do pierwszej niewciętej linii poza pętlą.

s

n

o

r

k

e

l

Done

Ciąg nie musi być ciągiem dosłownym. Może to być nazwa dowolnej zmiennej zawierającej ciąg. Na przykład wypróbuj ten kod:

my_word = "snorkel"

for x in my_word:

print(x)

print("Done")

Wynik jest dokładnie taki sam. Jedyna różnica polega na tym, że w pętli for użyliśmy nazwy zmiennej, a nie ciągu znaków. Ale „wiedział”, że masz na myśli zawartość my_word, ponieważ my_word nie jest ujęte w cudzysłów.

s

n

o

r

k

e

l

Done

Przeglądanie listy

Lista w Pythonie to w zasadzie dowolna grupa elementów oddzielonych przecinkami w nawiasach kwadratowych. Możesz przejść przez taką listę bezpośrednio w pętli for lub przez zmienną zawierającą listę. Oto przykład przeglądania listy bez zmiennej:

for x in ["The", "rain", "in", "Spain"]:

print(x)

print("Done")

Ten rodzaj pętli powtarza się raz dla każdego elementu na liście. Zmienna (x w poprzednim przykładzie) pobiera swoją wartość z jednego elementu na liście, idąc od lewej do prawej. Tak więc uruchomienie poprzedniego kodu daje wynik, który widzisz na rysunku

```
for x in ["The", "rain", "in", "Spain"]:
    print(x)
print("Done")
```

```
The
rain
in
Spain
Done
```

Listę można również przypisać do zmiennej, a następnie użyć nazwy zmiennej w pętli for zamiast listy. Rysunek poniżej pokazuje przykład, w którym zmienna seven_dwarves ma przypisaną listę siedmiu imion. Zwróć uwagę, że lista jest zawarta w nawiasach kwadratowych. To one sprawiają, że Python traktuje go jak listę. Pętla for przechodzi następnie przez listę, wyświetlając imię jednego karła (jednego elementu na liście) przy każdym przejściu przez pętlę. Użyliśmy nazwy zmiennej karzeł zamiast x, ale ta nazwa może być dowolną prawidłową nazwą, którą chcesz. Mogliśmy użyć x, mała_osoba, nazwa_fikcyjnej_jednostki, goober_wocky lub cokolwiek innego, o ile nazwa w pierwszym wierszu odpowiada nazwie użytej w pętli for.

```
seven_dwarves = ["Happy", "Grumpy", "Sleepy", "Bashful", "Sneezy", "Doc", "Dopey"]
for dwarf in seven_dwarves:
    print(dwarf)
print("And Snow White too")
```

```
Happy
Grumpy
Sleepy
Bashful
Sneezy
Doc
Dopey
And Snow White too
```

Wyjście z pętli

Zazwyczaj chcesz, aby pętla przechodziła przez całą listę lub zakres elementów, ale możesz również wymusić wcześniejsze zatrzymanie pętli, jeśli zostanie spełniony jakiś warunek. Użyj instrukcji break wewnątrz instrukcji if, aby wymusić wcześniejsze zatrzymanie pętli. Składnia to:

for x w pozycji:

if warunek:

[Zrób to ...]

break

Zrób to

zrób to po zakończeniu pętli

Umieszczamy [do this ...] w nawiasach kwadratowych, ponieważ umieszczanie kodu nad Continue jest opcjonalne, nie jest wymagane. Powiedzmy, że ktoś zdał egzamin, a my chcemy przejrzeć jego odpowiedzi. Ale mamy regułę, która mówi, że jeśli pozostawiają odpowiedź pustą, oznaczamy ją jako

niekompletną i ignorujemy pozostałe pozycje na liście. Oto jeden, w którym odpowiedzi na wszystkie elementy (bez pustych miejsc):

```
answers = ["A", "C", "B", "D"]

for answer in answers:

if answer == "":

print("Incomplete")

break

print(answer)

print("Loop is done")
```

W wyniku drukowane są wszystkie cztery odpowiedzi:

```
A

B

C

D

Loop is done
```

Oto ten sam kod, ale trzecia pozycja na liście jest pusta, na co wskazuje pusty ciąg ””.

```
answers = ["A", "C", "", "D"]

for answer in answers:

if answer == "":

print("Incomplete")

break

print(answer)

print("Loop is done")
```

Oto wynik uruchomienia tego kodu:

```
A

C

Incomplete

Loop is done
```

Tak więc logika jest taka, że jeśli jest jakaś odpowiedź, kod nie jest wykonywany i pętla działa do końca. Jeśli jednak pętla napotka pustą odpowiedź, wypisuje Incomplete, a także „break” pętlę, przeskakując do pierwszej instrukcji poza pętlą (ostatni kod bez wcięcia), który mówi, że Loop is done.

Zapętlanie z continue

Możesz także użyć instrukcji continue w pętli, co jest rodzajem przeciwieństwa break. Podczas gdy break powoduje, że wykonanie kodu przeskakuje poza koniec pętli i zatrzymuje zapętlenie, continue sprawia, że przeskakuje z powrotem na początek pętli i kontynuuje od następnego elementu (to znaczy po elemencie, który wyzwolił kontynuację). Oto ten sam kod, co w poprzednim przykładzie, ale zamiast wykonywania przerwy, gdy trafi na pustą odpowiedź, kontynuuje z następną pozycją na liście:

```
answers = ["A", "C", "", "D"]

for answer in answers:

if answer == "":

print("Incomplete")

continue

print(answer)

print("Loop is done")
```

Dane wyjściowe tego kodu są następujące. Nie wypisuje pustej odpowiedzi, wypisuje Niekompletne, ale potem wraca i kontynuuje przeglądanie pozostałych elementów:

```
A

C

Incomplete

D

Loop is done
```

Zagnieżdżanie pętli

Zagnieżdżanie pętli jest w porządku. . . to znaczy umieścić pętle wewnątrz pętli. Tylko upewnij się, że masz prawidłowe wcięcia, ponieważ tylko wcięcia określają, w której pętli znajduje się wiersz kodu. Na przykład na rysunku zewnętrzna pętla przechodzi przez słowa Pierwszy, Drugi i Trzeci. Przy każdym przejściu przez pętlę wypisuje słowo, a następnie wypisuje liczby 1–3 (przez pętlę przez zakres i dodanie 1 do każdej wartości zakresu). Jak widać, pętle działają, ponieważ widzimy każde słowo na zewnętrznej liście, po którym następują cyfry 1–3. Koniec pętli to pierwsza niewcięta linia na dole, która nie jest drukowana, dopóki zewnętrzna pętla nie zakończy swojego procesu.

```
# Outer Loop
for outer in ["First", "Second", "Third"]:
    print(outer)
    # Inner Loop
    for inner in range(3):
        print(inner + 1)

print("Both loops are done")
#Out of both loops here
```

```
First
1
2
3
Second
1
2
3
Third
1
2
3
Both loops are done
```

Zapętlanie z while

Jako alternatywę dla pętli z for możesz używać pętli with while. Różnica jest subtelna. Dzięki for zazwyczaj otrzymujesz ustaloną liczbę pętli, jedną dla każdego elementu w zakresie lub jedną dla każdego elementu na liście. Z pętlą while pętla działa tak długo, jak (while) jakiś warunek jest spełniony. Oto podstawowa składnia:

while condition

Zrób to ...

Zrób to ...

zrób to po zakończeniu pętli

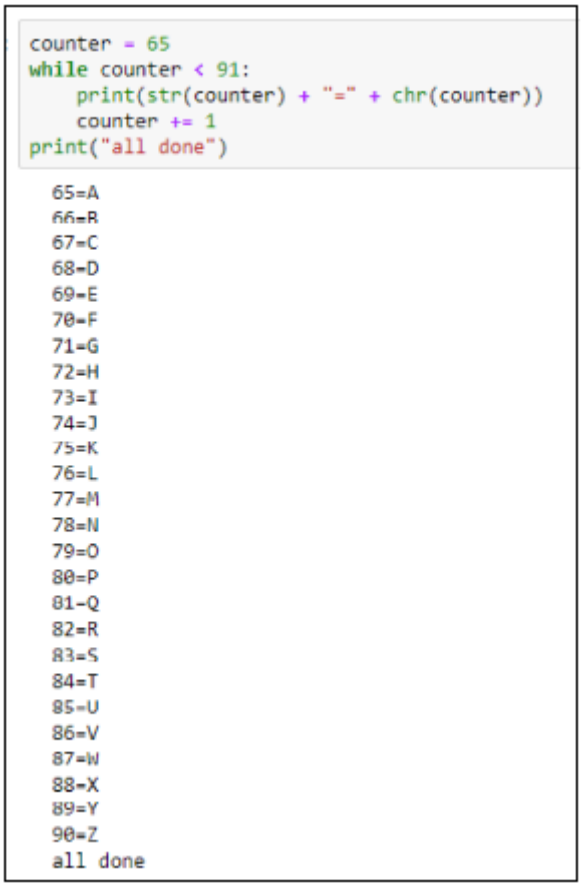
W przypadku pętli while musisz upewnić się, że warunek, który powoduje zatrzymanie pętli, w końcu się spełni. W przeciwnym razie otrzymujesz nieskończoną pętlę, która po prostu działa i działa i działa, dopóki jakiś błąd nie spowoduje awarii lub zmusisz ją do zatrzymania, zamykając aplikację, wyłączając komputer lub wykonując inną niezręczną rzecz. Oto przykład, w którym warunek while działa skończoną liczbę razy z powodu trzech rzeczy:

* Tworzymy zmienną o nazwie licznik i nadajemy jej wartość początkową (65).

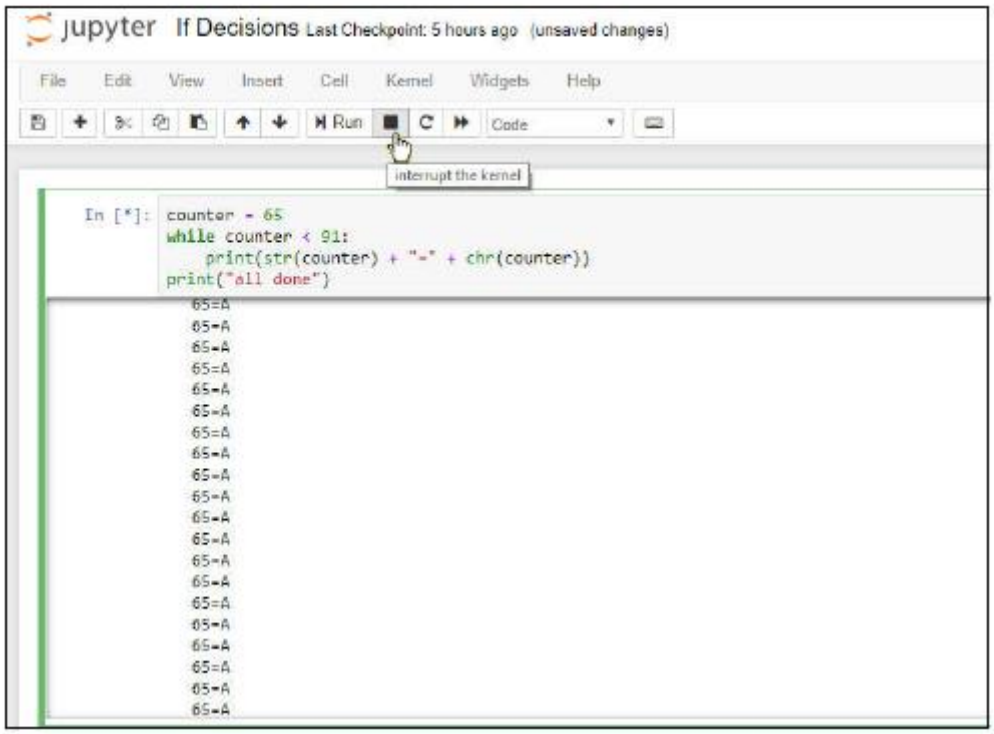
* Mówimy, aby uruchomić pętlę while , gdy licznik jest mniejszy niż 91.

* Wewnątrz pętli zwiększamy licznik o 1 (licznik = 1). To ostatecznie zwiększa licznik do ponad 91, co kończy pętlę.

Funkcja chr() wewnątrz pętli wyświetla po prostu znak ASCII dla dowolnej liczby w liczniku. Przejście od 65 do 90 wystarczy, aby wydrukować wszystkie wielkie litery alfabetu, jak widać na rysunku



Łatwym i powszechnym błędem, jaki można popełnić przy tego rodzaju pętli, jest zapomnienie o zwiększaniu licznika tak, aby rósł z każdym przejściem przez pętlę i ostatecznie powoduje, że warunek while staje się fałszywy i zatrzymuje pętlę. Na poniższym rysunku celowo usunęliśmy licznik = 1, aby spowodować ten błąd. Jak widać, nadal drukuje A. Trwa dłużej po tym, co widzisz na rysunku; musiałbyś przewinąć w dół, aby zobaczyć, ile zostało to zrobione do tej pory w dowolnym momencie. Jeśli zdarzy ci się to w notatniku Jupytera, nie panikuj. Po prostu naciśnij kwadratowy przycisk Stop po prawej stronie Run (pokazuje Przerwij jądro, co jest nerd = speak for stop ”po dotknięciu wskaźnika myszy). Powoduje to zatrzymanie wykonywania całego kodu w notesie. Następnie możesz kliknąć zakrzywioną strzałkę po prawej stronie przycisku Stop, aby ponownie uruchomić jądro i wrócić do punktu wyjścia. Następnie możesz naprawić błąd w kodzie i spróbować ponownie.



Rozpoczęcie pętli while z continue

Możesz użyć if i kontynuować w pętli while, aby przeskoczyć z powrotem na początek pętli, tak jak w przypadku pętli for. Spójrz na kod na rysunku jako przykład.

```
import random
print("Odd numbers")
counter = 0
while counter < 10:
    # Get a random number
    number = random.randint(1,999)
    if int(number / 2) == number /2:
        # If it's an even number, don't print it.
        continue
    #Otherwise, if it's odd, print it and increment the couter.
    print(number)
    # Increment the loop counter.
    counter += 1
print("Loop is done")
```

```
Odd numbers
697
449
91
567
949
333
591
699
[illegible]
837
Loop is done
```

Pętla while trwa, gdy zmienna o nazwie counter jest mniejsza niż 10. Wewnątrz pętli zmienna o nazwie number otrzymuje przypisaną do niej liczbę losową z zakresu od 1 do 999. Następnie to stwierdzenie:

if int(liczba / 2) == liczba / 2:

. . . sprawdza, czy liczba jest parzysta. Pamiętaj, że funkcja int() zwraca tylko całą część liczby. Powiedzmy, że losowa liczba, która zostanie wygenerowana, to 5. Dzielenie tej liczby przez 2 daje 2,5. Wtedy int(number) wynosi 2, ponieważ int() liczby usuwa wszystko po przecinku. 2 nie równa się 2,5, więc kod pomija kontynuację, wypisuje nieparzystą liczbę, zwiększa licznik i kontynuuje działanie. Jeśli następną liczbą losową jest powiedzmy 12; cóż, 12 podzielone przez 2 to 6, a int(6) równa się 6 (ponieważ żadna liczba nie ma kropki dziesiętnej). To powoduje, że kontynuuje wykonywanie, pomijając instrukcję print(number) i przyrost licznika, więc po prostu próbuje innej losowej liczby i kontynuuje swoją wesołą drogę. W końcu znajduje 10 liczb nieparzystych, w którym to momencie pętla się zatrzymuje, a ostatni wiersz kodu wyświetla „Loop is done".

Przerwanine pętli while z break

Możesz także przerwać pętlę while za pomocą break, tak jak w przypadku pętli for. Kiedy przerywasz pętlę while, wymuszasz kontynuowanie wykonywania od pierwszego wiersza kodu znajdującego się pod pętlą i poza nią, tym samym zatrzymując pętlę, ale kontynuując przepływ z resztą akcji po pętli. Innym sposobem myślenia o przerwie jest coś, co pozwala zatrzymać pętlę while, zanim warunek while okaże się fałszywy. Pozwala więc dosłownie wyrwać się z pętli przed czasem. Prawdę mówiąc, nie pamiętamy nawet sytuacji, w której wyłamanie się z pętli przed czasem było dobrym rozwiązaniem problemu, więc trudno podać praktyczny przykład. Zamiast tego pokażemy tylko składnię i podamy ogólny przykład. Składnia wygląda tak:

while condition1:

zrób to.

...

if warunek 2

break

zrób ten kod po zakończeniu pętli

Więc w zasadzie są dwie rzeczy, które mogą zatrzymać tę pętlę. Albo warunek 1 okaże się fałszem, albo warunek 2 okaże się prawdą. Niezależnie od tego, która z tych dwóch rzeczy się wydarzy, wykonanie kodu zostanie wznowione w pierwszym wierszu kodu poza pętlą, wiersz, który odczytuje, wykonuje ten kod, gdy pętla jest wykonana w przykładowej składni. Oto przykład, w którym program wypisuje do dziesięciu liczb, które można równomiernie podzielić przez pięć. Może jednak wydrukować mniej niż to, ponieważ gdy trafi na losową liczbę, która jest równo podzielna przez pięć, wyskakuje z pętli. Jedyną rzeczą, którą możesz przewidzieć, jest to, że wypisze od zera do dziesięciu liczb, które są równo podzielne przez pięć. Nie można przewidzieć, ile wydrukuje w danym przebiegu, ponieważ nie ma sposobu, aby stwierdzić, czy i kiedy otrzyma losową liczbę podzielną przez pięć podczas dziesięciu dozwolonych prób:

import random

print("Numbers that aren't evenly divisible by 5")

counter = 0

while counter < 10:

Get a random number

number = random.randint(1,999)

if int(number / 5) == number / 5:

If it's evenly divisible by 5, bail out.

break

Otherwise, print it and keep going for a while.

print(number)

Increment the loop counter.

counter = 1

print("Loop is done")

Tak więc przy pierwszym uruchomieniu tej aplikacji wynik może wyglądać podobnie do rysunku 2-14. Za drugim razem możesz otrzymać coś takiego jak na rysunku 2-15. Po prostu nie ma sposobu, aby przewidzieć, ponieważ liczba losowa jest rzeczywiście losowa i nieprzewidywalna (co jest ważną koncepcją w wielu grach). Tak więc przy pierwszym uruchomieniu tej aplikacji wynik może wyglądać podobnie do rysunku.

```
import random
print("Numbers that aren't evenly divisible by 5")
counter = 0
while counter < 10:
    # Get a random number
    number = random.randint(1,999)
    if int(number / 5) == number / 5:
        # If it's evenly divisble by 5, bail out.
        break
    #Otherwise, print it and keep going for a while.
    print(number)
    # Increment the loop counter.
    counter += 1
print("Loop is done")
```

```
Numbers that aren't evenly divisible by 5
729
754
317
753
327
366
69
813
543
67
Loop is done
```

Za drugim razem możesz otrzymać coś takiego jak na rysunku

```
import random
print("Numbers that aren't evenly divisible by 5")
counter = 0
while counter < 10:
    # Get a random number
    number = random.randint(1,999)
    if int(number / 5) == number / 5:
        # If it's evenly divisble by 5, bail out.
        break
    #Otherwise, print it and keep going for a while.
    print(number)
    # Increment the loop counter.
    counter += 1
print("Loop is done")
```

```
Numbers that aren't evenly divisible by 5
866
377
197
Loop is done
```

Po prostu nie ma sposobu, aby przewidzieć, ponieważ liczba losowa jest rzeczywiście losowa i nieprzewidywalna (co jest ważną koncepcją w wielu grach).