Jeśli uruchomimy ten kod, na ekranie uzyskamy piramidę gwiazdek:

```
*
  *
    *
      *
        *
          *
```


Jeśli ktoś poznał programowanie proceduralne, prawdopodobnie przygląda się wcześniejszemu kodowi i zastanawia się, co tu się dzieje, ponieważ języki proceduralne bardziej przypominają język angielski niż jest to w przypadku języków obiektowych. Instrukcja `Print "Hello World"` jest zgodna z formatem czasownik-obiekt, który jest zgodny z ogólną zasadą mówienia. Oderwijmy się na chwilę od programowania i przyjrzyjmy się konkretnemu przykładowi.

„Części mowy" języka VBA

Jeśli chcielibyśmy napisać kod dla instrukcji gry w piłkę za pomocą języka BASIC, instrukcja kopnięcia piłki mogłaby wyglądać następująco:

```
"Kick the Ball"
```

To tak samo, jak mówimy! I coś takiego ma sens. Mamy czasownik kopnij (kick), a następnie rzeczownik piłka (ball). Kod języka BASIC w poprzednim podrozdziale miał czasownik drukuj (Print) i rzeczownik gwiazdka (*). Życie jest piękne.

I tu mamy problem: język VBA nie działa w ten sposób. W rzeczywistości nie działa w ten sposób żaden język obiektowy. W języku obiektowym, obiekty (rzeczowniki) są najważniejsze, i stąd nazwa tych języków: obiektowe. Jeśli zamierzaliśmy napisać kod instrukcji do gry w piłkę za pomocą VBA, podstawowa struktura mogłaby mieć taką składnię:

```
Ball.Kick
```

Mamy rzeczownik piłka (`Ball`), który jest pierwszy. W języku VBA jest to *obiekt*. Następnie mamy czasownik kopnij (`Kick`). W języku VBA jest to *metoda*.

Podstawowa struktura języka VBA to seria wierszy kodu o poniższej składni:

```
Obiekt.Metoda
```

Nie trzeba mówić, że nie jest to język angielski. Jeśli w szkole posługiwaliśmy się romantycznym językiem, pamiętamy, że te języki korzystały z konstrukcji „rzeczownik-przymiotnik", ale nikt nie używał składni „rzeczownik-czasownik", prosząc kogoś, by coś zrobił:

```
woda.pić
jedzenie.jeść
dziewczyna.całować
```

I to jest przyczyna, dlaczego język VBA jest niezrozumiały dla osób, które poprzednio uczestniczyły w klasach programowania proceduralnego.

Rozwińmy analogię trochę bardziej. Wyobraźmy sobie, że chodzimy po trawiastym boisku, a przed nami widzimy pięć piłek. Jest to piłka do piłki nożnej (soccer), koszykówki, baseballa, kula do gry w kręgle i piłka do tenisa. Chcemy poinstruować dzieciaka w naszym zespole piłki nożnej, by kopnął piłkę (do piłki nożnej) „kick the soccer ball".

Jeśli powiemy kopnij piłkę (czyli `ball.kick`), nie będziemy mieli pewności, która z pięciu piłek zostanie kopnięta przez dziecko. Być może kopnie piłkę, która jest najbliżej, co może być problematyczne, jeśli będzie stał naprzeciw kuli do kręgli.

W przypadku większości rzeczowników, czyli obiektów w języku VBA, istnieje zbiór tego obiektu. Pomyślmy o programie Excel. Jeśli mamy jeden wiersz (row), możemy mieć kilka wierszy (rows). Jeśli mamy jedną komórkę (cell), możemy mieć grupę komórek (cells). Jeśli mamy jeden arkusz (worksheet), również możemy mieć wiele arkuszy (worksheets).

W języku angielskim, jedyna różnica pomiędzy obiektem a zbiorem tych obiektów polega na dodaniu litery s do nazwy obiektu:

`Row` zmienia się na `Rows`
`Cell` zmienia się na `Cells`
`Ball` zmienia się na `Balls`

Jeśli odwołujemy się do czegoś, co jest zbiorem (czy inaczej kolekcją), musimy poinformować język programowania, do którego elementu się odnosimy. Istnieje kilka metod zrealizowania tego zadania. Możemy odnosić się do elementu za pomocą liczb. Przykładowo, jeśli piłka nożna jest drugą w kolejności piłką, możemy „powiedzieć":

```
Balls(2).Kick
```

Metoda działa dobrze, ale może to być niebezpieczne dla programu. Na przykład, to dobrze zadziała we wtorek, ale jeśli przyjdziemy na boisko w środę i ktoś zmieni kolejność piłek, polecenie Balls(2).Kick może mieć bolesne konsekwencje.

Znacznie bezpieczniejsza metoda polega na używaniu nazw obiektów w kolekcji, która korzysta z takiej składni:

```
Balls("Soccer").Kick
```

Za pomocą tej metody zawsze wiemy, że kopnięta będzie piłka nożna.

Jak dotąd jest dobrze. Wiemy, że piłka będzie kopnięta i wiemy, że kopnięta będzie piłka nożna. Dla większości czasowników, czyli metod w języku Excel VBA, istnieją parametry, określające, jak ma być wykonane to działanie. Parametry te spełniają rolę przysłówków. Przykładowo, może chcieć, by piłka nożna była kopnięta silnie i w lewą stronę. W tym przypadku, metoda będzie miała kilka parametrów, które poinformują, w jaki sposób program powinien wykonać metodę (w poniższym przykładzie Direction oznacza kierunek, Force siłę, a Hard mocno):

```
Balls("Soccer").Kick Direction:=Left, Force:=Hard
```

W kodzie VBA, połączenie dwukropka i znaku równości (:=) wskazuje, że patrzymy na parametry, określające, jak powinna być wykonywana czynność (czyli czasownik).

Czasami metoda będzie miała listę 10 parametrów, a niektóre z nich są opcjonalne. Przykładowo, jeśli metoda `Kick` posiada parametr `Elevation` (wysokość), otrzymamy taki wiersz kodu:

```
Balls("Soccer").Kick Direction:=Left, Force:=Hard, Elevation:=High
```

W tym miejscu mamy pewne zagmatwanie: Każda metoda posiada domyślną kolejność swoich parametrów. Jeśli nie jesteśmy skrupulatnym programistą i znamy kolejność parametrów, możemy opuścić nazwy parametrów, jak w poniższym przykładzie, który jest równoważny poprzedniej linii kodu:

```
Balls("Soccer").Kick Left, Hard, High
```

Sposób ten nie ułatwia zrozumienia kodu. Bez znaków `:=` nie jest oczywiste, że są to parametry. Jeśli nie znamy kolejności parametrów, możemy nie zrozumieć tego zapisu. Dość prosto zrozumieć zapis w przypadku `Left`, `Hard` i `High`, ale jeśli są to następujące parametry:

```
ActiveSheet.Shapes.AddShape type:=1, Left:=10, Top:=20, _
    Width:=100, Height:=200
```

poniższy „skrótowy” zapis nie będzie zrozumiały:

```
ActiveSheet.Shapes.AddShape 1, 10, 20, 100, 200
```

Poprzedni wiersz kodu jest poprawny. O ile jednak nie znamy domyślnej kolejności parametrów tej metody `Add`, czyli `Type`, `Left`, `Top`, `Width` i `Height`, kod ten nie ma sensu. Domyślna kolejność parametrów danej metody to kolejność pokazywana w temacie Pomocy opisującym tę metodę.

Żeby jeszcze bardziej skomplikować życie, możemy rozpocząć specyfikowanie parametrów w ich porządku domyślnym bez nazywania parametrów, a następnie możemy rozpocząć nazywanie parametrów, jeśli akurat użyjemy parametru, który nie spełnia domyślnej kolejności. Przykładowo, jeśli chcemy kopnąć piłkę w lewo (left) i wysoko (high), ale nie ma znaczenia dla nas siła kopnięcia (czyli chcemy użyć wartości domyślnej dla parametru siły), poniższe dwie instrukcje mają identyczne działanie:

```
Balls("Soccer").Kick Direction:=Left, Elevation:=High
Balls("Soccer").Kick Left, Elevation:=High
```

Warto jednak pamiętać, że jeśli rozpoczniemy nazywanie parametrów, muszą one być również nazywane w pozostałej części wiersza.

Niektóre metody działają na swój własny sposób. Aby zasymulować naciśnięcie klawisza F9, używamy tego kodu:

```
Application.Calculate
```

Inne metody wykonują działanie i coś tworzą. Na przykład, za pomocą poniższego kodu możemy dodać arkusz:

```
Worksheets.Add Before:=Worksheets(1)
```

Ponieważ jednak metoda `Worksheets.Add` tworzy nowy obiekt, możemy do zmiennej przypisać wynik działania tej metody. W tym przypadku, musimy ująć parametry nawiasami:

```
Set MyWorksheet = Worksheets.Add(Before:=Worksheets(1))
```


Potrzebne jest jeszcze poznanie ostatniego fragmentu gramatyki: przymiotników. Podobnie jak przymiotnik opisuje rzeczownik, *właściwości* opisują obiekt. Ponieważ jesteśmy fanami programu Excel, przejdziemy z analogii piłkarskich do analogii z programem Excel. Istnieje obiekt do opisywania komórki aktywnej i na szczęście ma intuicyjną nazwę:

```
ActiveCell
```

Załóżmy, że chcemy zmienić kolor komórki aktywnej na czerwony. Istnieje właściwość nazwana `Interior.Color` dla komórki, która używa złożonego kodu. Możemy jednak zmienić kolor komórki na czerwony za pomocą poniższego kodu:

```
ActiveCell.Interior.Color = 255
```

Widzimy, jak może to być myląće. Mamy poprzednią konstrukcję *rzeczownik*-kropka-*coś*, ale tym razem jest to raczej `Obiekt.Właściwość`, a nie `Obiekt.Metoda`. Sposób rozróżniania tego jest dość subtelny: Przed znakiem równości brak jest dwukropka. Prawie zawsze wartość właściwości jest ustawiana tak samo jak coś innego lub inaczej mówiąc wartości właściwości przypisywana jest inna wartość (stąd znak równości).

Aby kolor tej komórki był taki sam, jak dla komórki A1, możemy „powiedzieć":

```
ActiveCell.Interior.Color = Range("A1").Interior.Color
```

`Interior.Color` to właściwość. Poprzez zmianę wartości właściwości zmieniamy wygląd rzeczy. Trochę to dziwne: Zmieniając przymiotnik, tak naprawdę modyfikujemy komórkę. Ludzie powiedzieliby „Pokoloruj komórkę na czerwono", natomiast w języku VBA zdanie takie ma postać:

```
ActiveCell.Interior.Color = 255
```

W tabeli 2.1 zamieszczono podsumowanie „części mowy" języka VBA.

Tabela 2.1 Części języka programowania VBA

Składnik VBA	Analogia	Uwagi
Obiekt	Rzeczownik	Przykład: komórka lub arkusz.
Kolekcja	Rzeczownik w liczbie mnogiej	Zazwyczaj określa, który z obiektów: `Sheets(1)`.
Metoda	Czasownik	Występuje w postaci: `Obiekt.Metoda`.
Parametr	Przysłówek	Lista parametrów po metodzie. Nazwa parametru oddzielana jest od wartości za pomocą znaków `:=`
Właściwość	Przymiotnik	Można ustawić właściwość (na przykład activecell.height=10) lub zapisać wartość właściwości (na przykład x = activecell.height).