The ByteBaker

Computer Science isn't a science and it's not about computers

Switch-case statement in Python

Posted on November 3, 2008 by Shrutarshi Basu

This post is part of the Powerful Python series where I talk about features of the Python language that make the programmer's job easier. The Powerful Python page (http://bytebaker.com/2009/03/26/powerful-python/) contains links to more articles as well as a list of future articles.

In the months since this was written, I have received a number of comments and learned more about programming languages in general. I put some of the new things I learned into a revised post on the topic (http://bytebaker.com/2009/08/17/switch-case-statement-in-python-revisited/) which you might be interested in.

A switch-case statement is a useful programming language that lets you control the flow of the program based on the value of a variable or expression. In particular, if the variable or expression that you're testing has a number of different of possible values, you could execute a block of code for each separate value. Here's a simple example in C (courtesy of Wikipedia):

```
1  switch(n) {
2    case 0:
3      printf("You typed zero.\n");
4      break;
5    case 1:
6    case 9:
7      printf("n is a perfect square\n");
8      break;
9    case 2:
10     printf("n is an even number\n");
11   case 3:
12   case 5:
13   case 7:
14     printf("n is a prime number\n");
15     break;
16
```

```
    case 4:
      printf("n is a perfect square\n");
    case 6:
    case 8:
      printf("n is an even number\n");
      break;
    default:
      printf("Only single-digit numbers are allowed\n");
    break;
}
```

Based on the value of the variable n, a different message will show up on the standard output. The `default` case is executed when the value of the variable doesn't match anything for which a case is defined. Notice that some cases have a break statement while others don't. If a particular case block is executed and no break is found, then all the following case blocks will be executed until a break is found.

The switch-case statement comes in handy when you're writing a language parser like I am now. For simple languages, like most programming languages, each token in a sample of the language can be followed only by a very limited number of possible tokens. My putting the options of possible following tokens in a switch case statement, you have an easy mechanism for performing different tasks based on what token is actually found. For example, if it's a variable name, you check if its been declared before, if it's an operator, perform the corresponding operation etc.

Unfortunately, my language of choice for the time being is Python, which doesn't come with a typical switch-case statement. One simple substitute is using a string of if-else blocks, with each if condition being what would have been a matching case. Here is part of the above code in Python using if-else blocks.

```
if n == 0:
    print "You typed zero.\n"
elif n== 1 or n == 9 or n == 4:
    print "n is a perfect square\n"
elif n == 2:
    print "n is an even number\n"
elif  n== 3 or n == 5 or n == 7:
    print "n is a prime number\n"
```

It certainly works and should be pretty easy to work, but it's not a very elegant solution. Especially if you have more than a handful of cases, you're going to have a very long string if-else cases. Since each of the if conditions must actually be checked, you might run into performance issues if this is a vital part of your code.

The Pythonic solution is to make use of Python's powerful dictionaries. Also known as associative arrays (http://en.wikipedia.org/wiki/Associative_array) in some languages, Python's dictionaries allow a simple one-to-one matching of a key and a value. When used as part of a switch case statement, the keys are what would normally trigger the case blocks. The interesting part is that the values in the dictionaries refer to functions that contain the code that would normally be inside the case blocks. Here's the above code rewritten as a dictionary and functions:

```
options = {0 : zero,
           1 : sqr,
```

```
                    4 : sqr,
                    9 : sqr,
                    2 : even,
                    3 : prime,
                    5 : prime,
                    7 : prime,
}

def zero():
    print "You typed zero.\n"

def sqr():
    print "n is a perfect square\n"

def even():
    print "n is an even number\n"

def prime():
    print "n is a prime number\n"
```

Now that you have the switch case setup, you can actually use it by simply doing a dictionary lookup:

```
options[num]()
```

Thanks to the fact that Python functions are first class values, you can use the functions as the values of the dictionary and then call them via dictionary lookup.

The advantage of this method is that it is generally cleaner than a long line of if-elses and someone reading your code can simply ignore the functions they are not interested in. Performance-wise, the Python dictionary lookup will almost certainly be more efficient than any solution you can rig yourself (with perhaps the exception of custom C code, but if you can do that, why are you reading this?). However, except a penalty associated with calling and coming back from a function. In fact, it's the function call that is both the strength and the weakness of this method. While it lets you cleanly segment your code, you also have a bunch of functions lying around. This isn't a problem if you were going to use functions anyways, but they can be a mess if you have a lot of tiny ones lying around. If your functions sufficiently small, consider inlining them in the dictionary as lambda (http://www.python.org/doc/2.5.2/tut/node6.html#SECTION006750000000000000000) expressions.

As a final note, the above example doesn't provide a default case in case nothing matches. You can make up for this by having the actual lookup be inside an if-else block, with the condition checking for the presence of the key in the dictionary (using the '<keyname> in <dictionary>' idiom). But a more Pythonic way is to wrap it in a try/except (http://docs.python.org/tut/node10.html) block. If the option isn't presence, a KeyError Exception will be raised which can then be caught and the default code executed. Be sure to check the Wikipedia (http://en.wikipedia.org/wiki/Switch_statement) entry and this blog post (http://stevemorin.blogspot.com/2005/11/python-switch-case-statement.html) for more information.

This entry was posted in Programming, Software and tagged functions, lambda, Python, switch-case. Bookmark the permalink.

43 thoughts on “Switch-case statement in Python”

Paddy3118 says:
on November 3, 2008 at 2:11 pm
Hi,
You gould also handle the missing default case by doing this instead:

options.get(num, lambda : None)()

Which will return None if num is not in options.

- Paddy.

Reply

> Kevin says:
> on October 9, 2013 at 12:03 pm
> but wouldn’t it return ‘None()’ if ‘num’ is not in options?
>
> Reply
> Kevin says:
> on October 9, 2013 at 12:10 pm
> Wouldn’t this be a better solution?
> options = {
> 0 : zero,
> 1 : sqr,
> 4 : sqr,
> 9 : sqr,
> 2 : even,

```
3 : prime,
5 : prime,
7 : prime,
}
try:
num = 'some mutable value'
options[num]()
except KeyError:
process_default()
```

Reply

Aruni says:
on November 5, 2008 at 8:34 am
Was in fact wondering how to avoid that if-else laddder. One of the practice programs I had done when first learning Python was a calculator – I believe that is in for an improvement.

Reply

mark says:
on January 12, 2009 at 6:57 am

The easiest solution would be to use ruby's case

```
x = "bla"

case x
when /some_regex/
puts "blablabla"
when 'a','b','c'
puts "yadda blablabla"
else
puts "some else blablabla"
end
```

The advantage would be to chain multiple if constructs together easily, without the cumbersome "or" chaining in the python example.

I think using functions via def is not a good way simply because a function should be somewhat "meaningful" in the first place, and not a substitute for no case/switch way control structures.

But then again using case/switch would violate python's principle of using as few idioms as possible, so as to not confuse the average programmer.

Reply

mark says:
on January 12, 2009 at 6:59 am

PS: The blog form doesnt like my 2 space indent
You could need some html tags to denote example code in those formulars though

Reply

beavis says:
on April 22, 2009 at 7:40 pm

That python doesn't have a switch statement is a huge black mark on the language and, frankly is an absolutely stupid decision. The example you used is too simplistic to point out how the ugly python hack fails.

```
switch(n) {
02. case 0:
03. printf("You typed zero.\n");
04. break;
05. case 1:
06. case 9:
07. printf("n is a perfect square\n");
08. break;
09. case 2:
10. printf("n is an even number\n");
11. case 3: printf("n is this");
12. case 5: printf("n is also this"
13. case 7:printf("n is too complex for silly python hacks")
14. printf("n is a prime number\n");
15. break;
16. case 4:
17. printf("n is a perfect square\n");
18. case 6:
19. case 8:
20. printf("n is an even number\n");
21. break;
22. default:
23. printf("Only single-digit numbers are allowed\n");
24. break;
25.}
```

Good luck using dictionary with this.

Switch statements are more powerful than any ugly python hack.

"But then again using case/switch would violate python's principle of using as few idioms as possible, so as to not confuse the average programmer."

If switch statements are confusing, then that "programmer" should find a new field.

Reply

Personoidon says:
on December 4, 2011 at 1:43 pm

options = {}
options[0] = { ‘cmd’: printf, ‘args’: “You typed zero.\n” }
options[1] = { ‘cmd’: printf, ‘args’: “n is a perfect square\n” }
options[9] = { ‘cmd’: printf, ‘args’: “n is a perfect square\n” }
options[7] = { ‘cmd’: printn, ‘args’: None }
if options.has_key(n):
options[n]['cmd'](options[n]['args'])
else:
printf(“Only single-digit numbers are allowed\n”);

???????
I’m pretty sure this would work.
If not, convert n to a string and use ‘n’ vs n as key.
I do want a switch statement, and I don’t want to say that this solves all of the problems, but you can get exactly the same functionality you just mentioned from python with dictionaries.
It’s nowhere near ideal, I just find your example to be nonsensical. I also can’t think of a situation where this couldn’t be applied as a solution. You simply need to create a function for each case.
Perhaps you’re the one who is confused by an alien solution?
And yes, I realize this post is from 2009, but I felt the need to point this out for those who mistakenly agree with you.

Reply

jamieson says:
on December 29, 2012 at 8:35 pm
Almost! You probably need to just add an asterisk, as in: (*options[n]['args'])

andraxin says:
on July 21, 2009 at 11:24 am
You do realize that NONE of the python “realizations”
you gave produces the same output as the C example,
right? The output of the C code will include *all*
options for a particular number (e.g. 2 is BOTH prime
AND even, 9 is a square AND odd, etc.), while your
attempts all cover only one of them.

Reply

Mark says:
on May 5, 2011 at 5:05 pm
Actually, if you compile (or read) the C code you will see that it does not.

Reply

Tz says:
on May 25, 2011 at 10:05 am
Actually, if you understand the C code you will see that it does.
There is no break statement after the tests for 2 and 4.

niet says:

on July 21, 2012 at 9:44 pm

Well, no, because it wouldn't compile (syntax error on line 12)

Shrutarshi Basu says:
on July 21, 2009 at 11:31 am
@andraxin: I had honestly not realized that before. You're quite right, and in retrospect, it's quite a drawback to the Python implementations I presented. I think this post needs a followup

Reply

Gary Feierbach says:
on October 12, 2013 at 10:35 pm
The must be another reason that the case statement was left out of Python. That it is not understandable by novices is a ridiculous reason. There are many other thing in Python that are far more challenging for a novice programmer. My 6 year old granddaughter immediately grasped the case statement but has a lot of trouble understanding implicit typing. Little wonder.

Reply

Mark Lawrence says:
on December 28, 2013 at 6:48 pm
Summed up here http://www.python.org/dev/peps/pep-3103/#rejection-notice

Kalifer Deil says:
on January 20, 2014 at 12:00 am
Thank you for sending me this. It just brings up more questions. When the case statement alternatives were presented did those in the group really understand the issues. I know many python programmers that came by way of HTML and Javascript rather than from a C background. The Javascript switch statement is the same as the C switch but I've rarely used it. Perhaps the context that requires Javascript code generally doesn't require that functionality. Maybe many of those that came to Python by that route don't really see the purpose. Certainly, if it is added to Python it would have to follow Python's syntax.

8. Pingback: Switch-case statement in Python revisited « The ByteBaker

9. Pingback: Powerful Python « The ByteBaker

10. Pingback: Revamping the ByteBaker series « The ByteBaker

regeya says:
on November 29, 2010 at 11:40 pm
if n == 0:
print "You typed zero."
elif n in (1, 4, 9):
print "n is a perfect square"
elif n == 2:
print "n is an even number"
elif n in (3, 5, 7):
print "n is a prime number"

Reply
elp says:
on December 6, 2010 at 9:57 am
Hi,
If you're searching extra-statement, as "switch", I built a python module that extends Python. It's called ESPY as Enhanced

Structure for Python and it's available for both Python 2.x and Python 3.x.
For example, in this case, a switch statement could be performed by the following code:
“””
macro switch(arg1):
….while True:
…….cont=False
……..val=%arg1%
……..socket case(arg2):
……if val==%arg2% or cont:
…….cont=True
…….socket
……..socket else:
……socket
……..break
“””
that can be used like this:
“””
a=3
switch(a):
….case(0):
……..print(“Zero”)
….case(1):
……..print(“Smaller than 2″):
……..break
….else:
……..print (“greater than 1″)
“””
so espy translate it in Python as:
“””
a=3
while True:
….cont=False
….if a==0 or cont:
……..cont=True
……..print (“Zero”)
….if a==1 or cont:
……..cont=True
……..print (“Smaller than 2″)
……..break

….print (“greater than 1″)
….break
‘‘’’’

You can find and test ESPY in this page:
elp.chronocv.fr/?lng=en

Eric LE PAPE

Reply
LobsterMan says:
on January 25, 2011 at 2:18 am
I was trying something similar, and it appears that you need to define the functions before you define the dictionary that will call those functions…

Reply
toto says:
on August 18, 2011 at 4:06 am
All time, I am searching about how to use switch in Ptyhon. So I can figured how to use with other function, because Python dont have switch function. Thx for sharing

Reply
Quincy says:
on May 15, 2012 at 12:21 pm
You wouldn’t have to define other functions if you used lambda. So one of the dict entries could be

0: lambda : print “This was zero”,

now you don’t have to have extra functions and such. (and it reads relatively similarly to what you had already).

if you wanted to change a variable (or variables) you could define a single function to do so and then call it with a lambda, e.g.

def change_foundoption(option):
global found_option
found_option = option

and then just do the following

0: lambda: change_foundoption(“none”)

etc

Reply
Quincy says:
on May 15, 2012 at 12:22 pm
which would, in fact, make it the equivalent of the ruby when/do statement, I believe

Reply

niet says:
on July 21, 2012 at 9:39 pm

```
if n == 0:
print("You typed zero.\n")
elif n in [1, 9]:
print("n is a perfect square\n")
elif n in [2, 3, 5, 7]:
if n == 2: print("n is an even number\n")
if n == 3: print("n is this")
if n == 5: print("n is also this")
if n == 7: print("n is too complex for silly python hacks")
printf("n is a prime number\n")
elif n in [4, 6, 8]:
if n == 4: print("n is a perfect square\n")
print("n is an even number\n")
else:
printf("Only single-digit numbers are allowed\n")
```

I don't think I've ever missed the switch statement, but I would not mind it either.

Reply

AKE says:
on August 12, 2012 at 3:51 am

@Shrutarshi: Actually, the comment by @andraxin illustrates precisely *why* C style switch statements are not a desirable element in Python. The point of Python is that there should be no ambiguity in code. The intent should be crystal clear from code by inspection. The example by @regeya achieves the desired behaviour and in a manner which makes the intent clear immediately. Contrast this with the C code snippet, and only a very close inspection of the code would reveal the programmer's intent.

C is designed for maximum flexibility — i.e. a hardware independent assembly language. Python is exactly the opposite: a language for programming clarity and ease.

Python should not duplicate every coding possibility in other languages, nor every 'cleverness' — most of these turn out to make code less immediately clear to a reader.

Reply

legine says:
on February 9, 2013 at 9:13 am

The suggested approch is very nice. Most people tend to think in switch case or if else construct. The suggested code uses Pythons design to make use of the Datastructure facility. Very nice approach.

Dont get locked into the reconstruction of C idioms, think in Problems and solutions. I had to fix a code once which had a if -else construction with 50 possible outcomes over 400 Lines. I had to draw a picture bevore I did understand what the writer wanted to accomplisch. A switch case would have eased the analyses a lot. (because it is always intended as a special case of if – else trees. Not only in C)

Have Fun!

Reply
Joe Milligan says:
on November 20, 2013 at 1:26 pm
I could not disagree more with this sentiment. Just last week, in a Java (shudder) class, I was writing a Hangman game and came up with this to draw the "man" based on wrong moves. I wrote the comment at the time, not in response to this thread.

(I'm just sure the formatting is going to get screwed up by the bbs, but it's still clear what is going on here.)

```
/* This demonstrates "fall through" at its finest. This is one of the reasons people turn away from
* Python, I think – there's no case-entry statement, and this technique has definite purposes. I've
* used this professionally in C on more than one occasion.
* In this case, we start with the final move (left leg) and "fall through" to the rest.
*/
switch( wrongMoves )
{
case 0:
break; // nothing else to draw
case 6: // left leg
g2d.draw( new Line2D.Float( personLLegCoords[0], personLLegCoords[1], personLLegCoords[2],
personLLegCoords[3] ) );
case 5: // right leg
g2d.draw( new Line2D.Float( personRLegCoords[0], personRLegCoords[1], personRLegCoords[2],
personRLegCoords[3] ) );
case 4: // left arm
g2d.draw( new Line2D.Float( personLArmCoords[0], personLArmCoords[1],
personLArmCoords[2], personLArmCoords[3] ) );
case 3: // right arm
g2d.draw( new Line2D.Float( personRArmCoords[0], personRArmCoords[1],
personRArmCoords[2], personRArmCoords[3] ) );
case 2: // body
g.fillRect( personBodyCoords[0], personBodyCoords[1], personBodyCoords[2],
personBodyCoords[3] );
case 1: // head
g.fillOval(personHeadCoords[0], personHeadCoords[1], personHeadCoords[2],
personHeadCoords[3] );
}
```

The only reason I wrote four lines of comments instead of just the last line is because this is code that was getting reviewed by fellow students whom I believe would have benefited from seeing a functional example of fall-through in a switch. It takes a massive tangle of spaghetti and condenses it into a neat little package.

Which is why I do this for a living. To figure out how to solve real problems. I don't do this to figure

out how I can get around fundamental deficiencies in a language (which is why I'm probably giving up on Java after this class).

I agree with Beavis: if switch statements are confusing, then leave it to those of us who can handle them and go do something more productive with your time. (I'll be happy to collect the paycheck you were too confused to earn.)

I'll go a step further and state that if you think dictionaries of functions are more clear to the reader than a time-tested switch statement, then I'll rewrite my Hangman program to draw just your head and the other part of your body it's so obviously inside.

Reply

Kalifer Deil says:
on November 24, 2013 at 10:20 pm
I agree Joe. I've used switch statements for building assemblers and for embedded applications for command interpreters. I would love to use python but the absence of a switch statement has put me off. I see switch statements as very easy to read and understand compared to using an elif block or using a mapping to a series of functions. Both of these alternatives to the switch statement are generally less efficient and more obscure.

I

Fabien says:
on November 25, 2013 at 4:11 am
Did you really need to be that aggressive ?

Claude Pache says:
on February 12, 2014 at 6:34 am
In your example, you could also write:

```
if (wrongMoves >= 6) {
// left leg
g2d.draw( … );
}
if (wrongMoves >= 5) {
// right leg
g2d.draw( … );
}
// etc.
```

which is not less clear than the `switch` statement.
From the perspective of clarity of code, I think that the only real advantage of `switch` statement, is that the expression being considered appears prominently once at the top the construct instead of being repeated over and over on each `if` clause. But it may be alleviated; for example:

```
result = some complex expression.
if (some condition on result) {
….
```

}
// etc.
Also, instead of writing `n == 1 or n == 4 or n == 9`, use `n in (1, 4, 9)`.

aaron says:
on March 12, 2014 at 4:31 am
You sound like you're hung up on functional programming, and haven't yet grasped the purpose of object oriented programming. Of course you hate Java, it's a very poor functional language, just like others hate C for not being object oriented. Take a decent structured programming class instead of a class focusing on a specific language. Every language has it's paradigm, and doing things in an objective way that nullifies the need for this extremely clunky kind of case statement on the hangman code is part of what Python is about.

Fabien says:
on March 14, 2014 at 12:30 pm
Mmh, by the way "This demonstrates "fall through" at its finest". Indeed :
1) First wrong move, you draw the head
2) Second wrong move, you draw the body, and the head
3) Third wrong move, you draw the right arm, the body, and the head
…

At the final wrong move, you will have drawn the head 6 times, the body 5 times, etc. Not what I call an efficient code…

8. Pingback: python - Simulating 'else' in dictionary switch statements | BlogoSfera

Francisco Luz says:
on March 30, 2013 at 6:38 pm
Here is another way to run the default value:
# define a default function
def defaultfunction():
print "this is the default value\n"

options.get(2, defaultfunction)()
# prints 'n is an even number'

options.get(8, defaultfunction)()
# prints 'this is the default value.'

Reply
renkotaro says:
on April 5, 2013 at 10:28 pm
thanks, it's really helped me….

Reply
Fabien says:
on October 7, 2013 at 5:41 am

```
zero = lambda: print("You typed zero.")
sqr = lambda: print("n is a perfect square")
even = lambda: print("n is an even number")
prime = lambda: print("n is a prime number")
default = lambda: print("only one digit number allowed")

options = {0 : (zero, sqr),
1 : (sqr,),
2 : (prime, even),
3 : (prime,),
4 : (sqr, even),
5 : (prime,),
6 : (even,),
7 : (prime,),
8 : (even,),
9 : (sqr,)
}

try:
for option in options[num]:
option()
except KeyError:
default()
```

Reply

Leif says:
on January 30, 2014 at 7:08 pm

I don't like all the ugly hacks for a switch statement. There's pretty sound reasoning for not having a switch statement in Python. The main being that there is no added performance benefit of having one, unlike C using a jump table. Also, switch statements cause repetitive code and is a sign of poor design. See http://c2.com/cgi/wiki?SwitchStatementsSmell.

I summed it up in http://allofthethings.org/missing-in-python/#Python_does_not_have_a_switch_statement

TLDR; Use a dictionary if an if-else is too long or requires a non-linear lookup. If that fails, rethink your design using the single responsibility principle.

I'd like someone to come to me with a circumstance that warrants a switch statement that isn't poor design because I can't think of any, and I couldn't think of any for my example.

Reply

23. Pingback: Python: switch case | Harry Tran

ralienpp says:
on March 13, 2014 at 9:59 am

I wonder why you say it is not efficient – implying that it goes into every branch. However that is not what I observe, have a look here:

```
def f4():
____x = 6
____if x == 6:
_________print 'old ', x
_________x = 7
_________print 'new ', x
____elif x == 7:
_________print 'again ', x
____elif x == 6:
_________print 'this cannot happen'
```

On my system I get this:
old 6
new 7

While if your statements were true, I should have seen
old 6
new 7
again 7

Reply

Ronald Duncan says:
on May 22, 2014 at 3:31 am
wow!
I read through the reject reason, and given that Python is implemented in C had difficulty understanding why case is difficult to implement in the underlying C given that almost all other languages have managed to create their own versions.
Back in 1989 I wrote a sales forecasting system in a spreadsheet, that had a macro language that supported cases. It made the problem trivial and I solved it in a few days. I was allowed 16k of code per cell, it was incredibly powerful.
Love Python but certain decisions such as single threading the engine and no case are part of the delights of having a Benevolent Dictator for Life, the upside is that it keeps is purity.
The lambda implementation of case really made me laugh, since how many python programmers have managed to get their heads around lambda – great powerful feature – and a nice example of it, but as an alternative to case

Reply