Replacements for switch statement in Python?

Ask Question

I want to write a function in Python that returns different fixed values based on the value of an input index.

In other languages I would use a `switch` or `case` statement, but Python does not appear to have a `switch` statement. What are the recommended Python solutions in this scenario?

python

edited Jan 20 at 19:41

Jerry Stratton
2,265 15 21

asked Sep 13 '08 at 0:36

Michael Schneider
7,497 3 16 13

63 Related PEP, authored by Guido himself: PEP 3103 – chb Jun 16 '12 at 17:22

20 @chb In that PEP, Guido doesn't mention that if/elif chains are also a classic source of error. It's a very fragile construct. – itsbruce

loss than performance or the fallthrough feature. – BobStein-VisiBone Oct 17 '14 at 19:04

19 All provided solutions are really ugly... switch case is so clean when reading. Cannot understand why it is not implemented in Python. – jmcollin92 Apr 8 '16 at 13:25

4 `switch` is actually more "versatile" than something returning different fixed values based on the value of an input index. It allows for different pieces of code to be executed. It actually does not even need to return a value. I wonder if some of the answers here are good replacements for a general `switch` statement, or only for the case of returning values with no possibility of executing general pieces of code. – sancho.s May 14 '17 at 21:29

47 Answers

1 2 next

You could use a dictionary:

```
def f(x):
    return {
        'a': 1,
        'b': 2,
    }[x]
```

answered Sep 13 '08 at 0:38

Greg Hewgill
626k 133 986 1140

Eli Bendersky Sep 19 '08 at 17:38

18 This is not a true switch/case... please see my response below – daniel Sep 20 '08 at 20:21

288 I'd recommend putting the dict outside of the function if performance is an issue, so it doesn't re-build the dict on every function call – Claudiu Oct 23 '08 at 16:22

34 @EliBendersky, Using the `get` method would probably be more normal than using a `collections.defaultdict` in this case. – Mike Graham Feb 23 '12 at 16:38

If you'd like defaults you could use the dictionary `get(key[, default])` method:

```
def f(x):
    return {
        'a': 1,
        'b': 2
    }.get(x, 9)    # 9 is de
```

edited Jul 6 '17 at 22:51

Nilpo

3,464 1 14 26

answered Sep 19 '08 at 15:45

John Mee Apr 9 '10 at 7:57

12 @JM: Well, obviously dictionary lookups don't support fall-throughs. You could do a double dictionary lookup. I.e. 'a' & 'b' point to answer1 and 'c' and 'd' point to answer2, which are contained in a second dictionary. – Nick Apr 9 '10 at 9:54

1 this is better to pass a default value – HaTiMSuM Oct 13 '16 at 11:01

There is a problems with this approach, first each time you call f you going to create the dict again second if you have more complex value you can get an exceptions ex. if x is a tuple and we are want to do something like this x = ('a') def f(x): return { 'a': x[0], 'b': x[1] }.get(x[0], 9) This will raise IndexError – Idan Haim Shalom Aug 30 '17 at 8:01

1 @Idan: The question was to replicate switch. I'm sure I could break this code too if I tried putting in odd values. Yes, it will recreate, but it is simple to fix. – Nick Sep 19 '17 at 21:59

I've always liked doing it this way

```
result = {
  'a': lambda x: x * 5,
  'b': lambda x: x + 7,
  'c': lambda x: x - 2
}[value](x)
```

From here

answered Sep 13 '08 at 0:41

Mark Biek
85.5k 50 141 192

9 great method, combined with get() to handle default is my best choice too – drAlberT Sep 2 '09 at 16:11

20 it maybe isn't a good idea to use lambda in this case because lambda is actually called each time the dictionary is built. – htmlfarmer Apr 22 '12 at 21:48

9 Sadly this is the closest people are going to get. Methods which use `.get()` (like the current highest answers) will need to eagerly evaluate all possibilities before dispatching, and therefore not only are (not just very but) extremely inefficient and also cannot have side-effects; this answer gets around that issue, but is more verbose. I would just use if/elif/else, and even those take just as long to write as 'case'. – ninjagecko Mar 17 '14 at 13:48

10 wouldn't this evaluate all the functions/lambdas every time in all cases, even if it is only returning one of the results? – slf Aug 6 '14 at 19:04

14 @slf No, when the control flow reaches that piece of code, it will build 3 functions (through the use of the 3 lambdas) and then build a dictionary with those 3 functions as values, but they remain uncalled (*evaluate* is

of the 3 keys). The function hasn't been called at that point, yet. Then `(x)` calls the just returned function with `x` as argument (and the result goes to `result`). The other 2 functions won't be called. – blubberdiblub Sep 18 '15 at 6:11

In addition to the dictionary methods (which I really like, BTW), you can also use if-elif-else to obtain the switch/case/default functionality:

```
if x == 'a':
    # Do the thing
elif x == 'b':
    # Do the other thing
if x in 'bc':
    # Fall-through by not us
elif x in 'xyz':
    # Do yet another thing
else:
    # Do the default
```

This of course is not identical to switch/case - you cannot have fall-through as easily as leaving off the break; statement, but you can have a more complicated test. Its formatting is nicer than a series of nested ifs, even though functionally that's what it is closer to.

edited Jun 10 '15 at 14:07

answered Sep 13 '08 at 1:10

throw a KeyError if no matching case is found – martyglaubitz May 18 '13 at 10:30

5 I thought about the dictionary / `get` way, but the standard way is simply more readable. – Martin Thoma Jun 25 '15 at 6:33

Doesn't exactly operate like a switch statement but very close. In my opinion closest thing – Arijoon Aug 26 '15 at 20:37

This is the cleanest solution. It's most common that each `switch` has a `break` and the most common use for fall-through that I see is to match multiple elements, as in `if x in 'bc': .` – bmacnaughton Dec 30 '15 at 16:32

3 Nice, the "Fall-through by not using elif " is a bit confusing, though. What about this: forget about "fall through" and just accept it as two `if/elif/else` 's? – Alois Mahdal May 30 '16 at 11:40

My favorite Python recipe for switch/case is:

```
choices = {'a': 1, 'b': 2}
result = choices.get(key, 'd
```

Short and simple for simple scenarios.

Compare to 11+ lines of C code:

```
    case 'b' :
        result = 2;
        break;
    default :
        result = -1;
}
```

You can even assign multiple variables by using tuples:

```
choices = {'a': (1, 2, 3), '
(result1, result2, result3)
```

edited Aug 16 '15 at 21:58

answered Jun 17 '15 at 2:25

ChaimG
1,811 1 13 28

10 I find this to be a more robust answer than the accepted. – cerd Aug 18 '15 at 23:16

Note that the 2 segments of code aren't the same. The python version returns 'default' when no match but the C version returns -1. When you replace the line as `choices.get(key, -1)` , it becomes harder to read because people have to pause to think how get() works and what `-1` mean. – some user Feb 29 '16 at 19:05

2 @some user: Personally, I find {}.get(,) readable. For extra readability for Python beginners you may want to use `default = -1; result = choices.get(key, default)` . – ChaimG Mar 1 '16 at 0:19

2 compare with 1 line of c++

`result = 1 if key == 'a' else (2 if key == 'b' else 'default')` . but is the one liner readable? – ChaimG Aug 24 '16 at 2:57

```
class switch(object):
    value = None
    def __new__(class_, valu
        class_.value = value
        return True

def case(*args):
    return any((arg == switc
```

Usage:

```
while switch(n):
    if case(0):
        print "You typed zer
        break
    if case(1, 4, 9):
        print "n is a perfec
        break
    if case(2):
        print "n is an even
    if case(2, 3, 5, 7):
        print "n is a prime
        break
    if case(6, 8):
        print "n is an even
        break
    print "Only single-digit
    break
```

Tests:

```
n = 2
#Result:
#n is an even number.
#n is a prime number.
n = 11
#Result:
#Only single-digit numbers a
```

edited May 31 '13 at 14:24

KurzedMetal
8,642 3 28 50

several switches are hit at the same time all switches take the value of the last switch. – francescortiz Jun 26 '13 at 16:35

30 While @francescortiz likely means thread safe, it's also not threat safe. It threatens the values of the variables! – Zizouz212 Jun 16 '15 at 16:29

4 The thread safety problem could likely be worked around by using thread-local storage. Or it could be avoided altogether by returning an instance and using that instance for the case comparisons. – blubberdiblub Sep 18 '15 at 6:24

3 @blubberdiblub But then isn't it just more efficient to use a standard `if` statement? – wizzwizz4 May 23 '16 at 17:32

3 This is also not safe if used in multiple functions. In the example given, if the `case(2)` block called another function that uses switch(), then when doing `case(2, 3, 5, 7)` etc to look for the next case to execute, it will use the switch value set by the other function not the one set by the current switch statement. – user9876 Aug 17 '17 at 8:58

There's a pattern that I learned from Twisted Python code.

```
class SMTP:
```

```
SMTP().lookupMethod('HELO')(
SMTP().lookupMethod('QUIT')(
```

You can use it any time you need to dispatch on a token and execute extended piece of code. In a state machine you would have `state_` methods, and dispatch on `self.state`. This switch can be cleanly extended by inheriting from base class and defining your own `do_` methods. Often times you won't even have `do_` methods in the base class.

Edit: how exactly is that used

In case of SMTP you will receive `HELO` from the wire. The relevant code (from `twisted/mail/smtp.py`, modified for our case) looks like this

```
class SMTP:
    # ...

    def do_UNKNOWN(self, res
        raise NotImplemented

    def state_COMMAND(self,
        line = line.strip()
        parts = line.split(N
        if parts:
            method = self.lo
            if len(parts) ==
                return metho
            else:
                return metho
        else:
            raise SyntaxErro

SMTP().state_COMMAND('   HEL
```

You'll receive `'  HELO foo.bar.com '` (or you might get `'QUIT'` or `'RCPT TO:` [illegible]). This is tokenized into

(The original method is also called `state_COMMAND`, because it uses the same pattern to implement a state machine, i.e. `getattr(self, 'state_' + self.mode)`)

edited Sep 13 '08 at 17:42

answered Sep 13 '08 at 1:26
user6205

I don't see the benefit from this pattern over just calling the methods directly: SMTP().do_HELO('foo.bar.com') OK, there can be common code in the lookupMethod, but since that also can be overwritten by the subclass I don't see what you gain from the indirection. – Mr Shark Sep 13 '08 at 11:35

1 You wouldn't know what method to call in advance, that is to say 'HELO' comes from a variable. i've added usage example to the original post – user6205 Sep 13 '08 at 17:45

May I suggest simply: eval('SMTP().do_' + command)('foo.bar.com') – jforberg Jun 21 '11 at 17:32

Also, why instantiate a new SMTP object for each method call? That's what global functions are for. – jforberg Jun 21 '11 at 17:34

7 eval? seriously? and instead of instantiating one

My favorite one is a really nice recipe. You'll really like it. It's the closest one I've seen to actual switch case statements, especially in features.

Here's an example:

```
# The following example is p
# but is included for its si
# in each suite.
v = 'ten'
for case in switch(v):
    if case('one'):
        print 1
        break
    if case('two'):
        print 2
        break
    if case('ten'):
        print 10
        break
    if case('eleven'):
        print 11
        break
    if case(): # default, co
        print "something els
        # No need to break h

# break is used here to look
# elif is generally just as

# Empty suites are considere
# should contain 'pass'
c = 'z'
for case in switch(c):
    if case('a'): pass # onl
    if case('b'): pass
    # ...
    if case('y'): pass
    if case('z'):
        print "c is lowercas
        break
    if case('A'): pass
    # ...
    if case('Z'):
        print "c is uppercas
        break
    if case(): # default
        print "I dunno what
```

```
if case(*string.lowercas
    print "c is lowercas
    break
if case(*string.uppercas
    print "c is uppercas
    break
if case('!', '?', '.'):
    print "c is a senten
    break
if case(): # default
    print "I dunno what
```

answered Jul 7 '11 at 6:12

John Doe
2,218 1 12 20

2 I would substitute `for case in switch()` with `with switch() as case` , makes more sense, since it need s to run only once. – Ski Dec 12 '13 at 16:24

2 @Skirmantas: Note that `with` doesn't allow for `break` though, so the fallthrough option is taken away. – Jonas Wielicki May 8 '14 at 16:53

5 Apologies for not putting more effort in to determine this myself: a similar answer above is not thread safe. Is this? – David Winiecki Sep 12 '14 at 15:47

1 @DavidWiniecki The code components missing from the above (and possibly copyright by activestate) appear to be thread safe. – Jasen Aug 23 '16 at 22:09

would another version of this be something like `if c in set(range(0,9)): print "digit" elif c in set(map(chr, range(ord('a'), ord('z')))): print "lowercase"` ? – mpag Oct 31 '16 at 19:08

```
    def __enter__(self): ret
    def __exit__(self, type,
    def __call__(self, *mcon

from datetime import datetim
with Switch(datetime.today()
    if case(0):
        # Basic usage of swi
        print("I hate monday
        # Note there is no b
    elif case(1,2):
        # This switch also s
        print("When is the w
    elif case(3,4): print("T
    else:
        # Default would occu
        print("Let's go have
```

edited Dec 15 '17 at 20:25

Tadhg McDonald-Jensen
11k 3 15 36

answered May 3 '15 at 9:05

Ian Bell
261 3 2

4 Using context managers is a good creative solution. I'd recommend adding a bit of explanation and maybe a link to some information on Context Managers to give this post some, well, context ;) – Will May 3 '15 at 9:13

2 I don't like if/elif chains much but this is both the most creative and the most practical of all the solutions I've seen using Python's existing syntax. – itsbruce Oct 2 '17 at 8:05

Let's say you don't want to just return a value, but want to use methods that change something on an object.

```
  'b': obj.decrement(x)
}.get(value, obj.default(x))
```

What happens here is that python evaluates all methods in the dictionary. So even if your value is 'a', the object will get incremented **and** decremented by x.

Solution:

```
func, args = {
  'a' : (obj.increment, (x,)
  'b' : (obj.decrement, (x,)
}.get(value, (obj.default, (

result = func(*args)
```

So you get a list containing a function and its arguments. This way, only the function pointer and the argument list get returned, *not* evaluated. 'result' then evaluates the returned function call.

answered Sep 30 '10 at 8:31

GeeF
432 1 5 13

expanding on the "dict as switch" idea. if you want to use a default value for your switch:

```
def f(x):
    try:
        return {
            'a': 1,
            'b': 2,
        }[x]
    except KeyError:
        return 'default'
```

11 I think it's clearer to use .get() on the dict with the default specified. I prefer to leave Exceptions for exceptional circumstances, and it cuts three lines of code and a level of indentation without being obscure. – Chris B. Jun 5 '09 at 15:14

9 This **is** an exceptional circumstance. It may or may not be a *rare* circumstance depending on useful, but it's definitely an exception (fall back on `'default'`) from the rule (get something from this dict). By design, Python programs use exceptions at the drop of a hat. That being said, using `get` could potentially make the code a bit nicer. – Mike Graham Mar 26 '10 at 16:49

If you have a complicated case block you can consider using a function dictionary lookup table...

If you haven't done this before its a good idea to step into your debugger and view exactly how the dictionary looks up each function.

NOTE: Do *not* use "()" inside the case/dictionary lookup or it will call each of your functions as the dictionary / case block is created. Remember this because you only want to

```
    print "second"

def third_case():
    print "third"

mycase = {
'first': first_case, #do not
'second': second_case, #do n
'third': third_case #do not
}
myfunc = mycase['first']
myfunc()
```

edited Oct 17 '12 at 21:52

answered Apr 22 '12 at 21:43

htmlfarmer
1,359 4 19 28

I'm just going to drop my two cents in here. The reason there isn't a case/switch statement in Python is because Python follows the principle of 'Theres only one right way to do something'. So obviously you could come up with various ways of recreating switch/case functionality, but the Pythonic way of accomplishing this is the if/elif construct. ie

```
if something:
    return "first thing"
elif somethingelse:
    return "second thing"
elif yetanotherthing:
    return "third thing"
else:
    return "default thing"
```

I just felt PEP 8 deserved a

our in PEP 8, including "There's only one right way to do something"

answered Aug 14 '17 at 22:19

user2233949
687 8 16

3 So why does Python have for loops and while loops? Everything you can do with a for loop you can implement with a while loop. – itsbruce Oct 1 '17 at 10:01

1 True. Switch/case are too often abused by beginning programmers. What they really want is the *strategy pattern*. – user228395 Oct 6 '17 at 12:34

Sounds like Python wishes it was Clojure – T.W.R. Cole Feb 16 at 5:17

@T.W.R.Cole I don't think so, Python was doing it first. Python has been around since 1990 and Clojure since 2007. – Taylor May 9 at 16:52

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:

```
                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
        break
    else:
        print ("greater than
```

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
        break
    print ("greater than 1")
    break
```

edited Nov 8 '11 at 10:25

bluish
12.8k 14 90 143

answered Dec 6 '10 at 15:08

elp
111 1 2

Very cool, but what's the point of the `while True:` at the top of the generated Python code? It'll inevitably hit the `break` at the bottom of the generated Python code, so it seems to me that both the `while True:` and `break` could be removed. Further, is ESPY smart

the-less. +1 for sheer coolness. – ArtOfWarfare Jun 29 '14 at 12:56

I found that a common switch structure:

```
switch ...parameter...
case p1: v1; break;
case p2: v2; break;
default: v3;
```

can be expressed in Python as follows:

```
(lambda x: v1 if p1(x) else
```

or formatted in a clearer way:

```
(lambda x:
     v1 if p1(x) else
     v2 if p2(x) else
     v3)
```

Instead of being a statement, the python version is an expression, which evaluates to a value.

edited May 29 '15 at 17:25

answered Jan 2 '15 at 18:01

leo
180 1 6

Also instead of ...parameter... and p1(x) how about `parameter` and `p1==parameter` – BobStein-VisiBone Mar 11 '15 at 15:28

@BobStein-VisiBone hi, here

denotes a predicate; as long as it returns `True` or `False`, no matter it is a function call or a expression, it's fine. – leo Mar 13 '15 at 16:16

@BobStein-VisiBone Yes, you are right! Thank :) For the multi-line expression to work, parentheses should be placed, either as in your suggestion, or as in my modified example. – leo Mar 14 '15 at 5:16

Excellent. Now I'll delete all my comments about the parens. – BobStein-VisiBone Mar 14 '15 at 20:48

I didn't find the simple answer I was looking for anywhere on Google search. But I figured it out anyway. It's really quite simple. Decided to post it, and maybe prevent a few less scratches on someone else's head. The key is simply "in" and tuples. Here is the switch statement behavior with fall-through, including RANDOM fall-through.

```
l = ['Dog', 'Cat', 'Bird', '
     'Dragonfly', 'Snake', '

for x in l:
    if x in ('Dog', 'Cat'):
        x += " has four legs
    elif x in ('Bat', 'Bird'
        x += " has wings."
    elif x in ('Snake',):
        x += " has a forked
    else:
        x += " is a big myst
    print(x)
```

```
        x = "Value 2 caught
    elif x in (3, 7, 8):
        x = "Values 3, 7, 8
    elif x in (4, 6):
        x = "Values 4 and 6
    else:
        x = "Values 5 and 9
    print(x)
```

Provides:

```
Dog has four legs
Cat has four legs
Bird has wings.
Bigfoot is a big mystery by
Dragonfly has wings.
Snake has a forked tongue.
Bat has wings.
Loch Ness Monster is a big m

Values 0 and 1 caught here.
Values 0 and 1 caught here.
Value 2 caught here.
Values 3, 7, 8 caught here.
Values 4 and 6 caught here
Values 5 and 9 caught in def
Values 4 and 6 caught here
Values 3, 7, 8 caught here.
Values 3, 7, 8 caught here.
Values 5 and 9 caught in def
```

edited Oct 12 '13 at 18:43

answered Oct 12 '13 at 15:04

JD Graham
111 1 3

Where exactly is fallthrough here? – Jonas Wielicki May 8 '14 at 16:56

switch(n){ case – JD Graham Jul 30 '14 at 4:46

Oops! There is fall through there, but I'm not contributing to Stack Overflow anymore. Don't like THEM at all. I like the contributions by others, but

break statement in a switch. – JD Graham Jul 30 '14 at 4:58

1 Here both the values "Dog" and "Cat" FALL THROUGH and are handled by the SAME functionality, which is they are defined as having "four legs." It's an ABSTRACT equivalent to fall through and different values handled by the SAME case statement where a break occurs. – JD Graham Jul 30 '14 at 5:16

@JDGraham I think Jonas meant another aspect of fallthrough, which happens when programmer occasionally forget to write `break` in the end of the code for a `case`. But I think we don't need *such* "fallthrough" :) – Mikhail Batcer Aug 12 '15 at 9:00

The solutions I use:

A combination of 2 of the solutions posted here, which is relatively easy to read and supports defaults.

```
result = {
  'a': lambda x: x * 5,
  'b': lambda x: x + 7,
  'c': lambda x: x - 2
}.get(whatToUse, lambda x: x
```

where

```
.get('c', lambda x: x - 22)(
```

looks up `"lambda x: x - 2"` in the dict and uses it with

`x - 22"` with `x=44` .

edited Feb 23 '12 at 16:27

bluish
12.8k 14 90 143

answered Jun 28 '10 at 1:32

thomasf1
822 3 13 26

sleek & elegant.. Thanks a lot... – S.K. Venkat Feb 20 at 19:11

```
# simple case alternative

some_value = 5.0

# this while loop block simu

# case
while True:

    # case 1
    if some_value > 5:
        print ('Greater than
        break

    # case 2
    if some_value == 5:
        print ('Equal to fiv
        break

    # else case 3
    print ( 'Must be less th
    break
```

edited Oct 24 '15 at 1:17

pbfy0
569 3 13

answered Aug 13 '15 at 17:40

user5224656
79 1 1

hmmm, is it duplicity answer

I have made a (relatively) flexible and re-usable solution for this. It can be found at GitHub as this gist. If the result of the switch function is callable, it is automatically called.

answered Jun 5 '09 at 14:20

I liked Mark Bies's answer

Since the `x` variable must used twice, I modified the lambda functions to parameterless.

I have to run with

```
results[value](value)
```

```
In [2]: result = {
    ...:   'a': lambda x: 'A
    ...:   'b': lambda x: 'B
    ...:   'c': lambda x: 'C
    ...: }
    ...: result['a']('a')
    ...:
Out[2]: 'A'

In [3]: result = {
    ...:   'a': lambda : 'A'
    ...:   'b': lambda : 'B'
    ...:   'c': lambda : 'C'
    ...:   None: lambda : 'N

    ...: }
    ...: result['a']()
    ...:
Out[3]: 'A'
```

Edit: I noticed that I can use `None` type with with dictionaries. So this would emulate `switch ; case else`

guneysus
3,593 1 26 28

Doesn't the None case emulate simply `result[None]()` ? – BobStein-VisiBone Mar 11 '15 at 15:19

Yes, exactly. I mean `result = {'a': 100, None:5000}; result[None]` – guneysus Mar 11 '15 at 15:52

2 Just checking that no one is thinking `None:` behaves like `default: .` – BobStein-VisiBone Mar 11 '15 at 16:54

```
def f(x):
    return 1 if x == 'a' el
           2 if x in 'bcd'
           0 #default
```

Short and easy to read, has a default value and supports expressions in both conditions and return values.

However, it is less efficient than the solution with a dictionary. For example, Python has to scan through all the conditions before returning the default value.

answered Nov 5 '12 at 20:05

emu
836 8 17

Defining:

allows you to use a fairly straightforward syntax, with the cases bundled into a map:

```
def sample1(x):
  local = 'betty'
  switch1(x, {
    'a': lambda: print("hell
    'b': lambda: (
      print("goodbye," + loc
      print("!")),
    })
```

I kept trying to redefine switch in a way that would let me get rid of the "lambda:", but gave up. Tweaking the definition:

```
def switch(value, *maps):
  options = {}
  for m in maps:
    options.update(m)
  if value in options:
    options[value]()
  elif None in options:
    options[None]()
```

Allowed me to map multiple cases to the same code, and to supply a default option:

```
def sample(x):
  switch(x, {
    _: lambda: print("other"
    for _ in 'cdef'
    }, {
    'a': lambda: print("hell
    'b': lambda: (
      print("goodbye,"),
      print("!")),
    None: lambda: print("I d
    })
```

Each replicated case has to be in its own dictionary; switch() consolidates the dictionaries before looking up the value. It's still uglier

answered Jul 25 '13 at 18:23

William H. Hooper
312 3 6

I think the best way is to **use the python language idioms to keep your code testable**. As showed in previous answers, I use dictionaries to **take advantage of python structures and language** and keep the "case" code isolated in different methods. Below there is a class, but you can use directly a module, globals and functions. The class has methods that **can be tested with isolation**. Dependending to your needs, you can play with static methods and attributes too.

```
class ChoiceManager:

    def __init__(self):
        self.__choice_table
        {
            "CHOICE1" : self
            "CHOICE2" : self
        }

    def my_func1(self, data)
        pass

    def my_func2(self, data)
        pass

    def process(self, case,
        return self.__choice

ChoiceManager().process("CHO
```

It is possible to **take advantage of this method**

Supposing you have to process a lot of messages or packets from the net or your MQ. Every packet has its own structure and its management code (in a generic way). With the above code it is possible to do something like this:

```
class PacketManager:

    def __init__(self):
        self.__choice_table
        {
            ControlMessage :
            DiagnosticMessag
        }

    def my_func1(self, data)
        # process the contro
        pass

    def my_func2(self, data)
        # process the diagno
        pass

    def process(self, pkt):
        return self.__choice

pkt = GetMyPacketFromNet()
PacketManager().process(pkt)


# isolated test or isolated
def test_control_packet():
    p = ControlMessage()
    PacketManager().my_func1
```

So **complexity is not spread in the code flow but it is rendered in code structure**.

edited Apr 20 '16 at 14:54

answered Mar 18 '16 at 8:01

J_Zar
907 9 23

Really ugly... switch case is so clean when reading. Cannot understand why it is not implemented in Python. – jmcollin92 Apr 8 '16 at 13:25

@AndyClifton: I'm sorry... an example? Think of every time you need to have multiple decision branching code and you can apply this method. – J_Zar Apr 11 '16 at 6:51

@jmcollin92: the switch statement is confortable, I agree. However the programmer tends to write very long statements and code which is not reusable. The way I described is cleaner to test and more reusable, IMHO. – J_Zar Apr 11 '16 at 6:53

1 @AndyClifton: I'm sorry, I'm late but I posted some example case. – J_Zar Apr 20 '16 at 14:56

Most of the answers here are pretty old, and especially the accepted ones, so it seems worth updating.

First, the official Python FAQ covers this, and recommends the `elif` chain for simple cases and the `dict` for larger or more complex cases. It also suggests a set of `visit_` methods (a style used by many server frameworks) for some cases:

The FAQ also mentions PEP 275, which was written to get an official once-and-for-all decision on adding C-style switch statements. But that PEP was actually deferred to Python 3, and it was only officially rejected as a separate proposal, PEP 3103. The answer was, of course, no—but the two PEPs have links to additional information if you're interested in the reasons or the history.

One thing that came up multiple times (and can be seen in PEP 275, even though it was cut out as an actual recommendation) is that if you're really bothered by having 8 lines of code to handle 4 cases, vs. the 6 lines you'd have in C or Bash, you can always write this:

```
if x == 1: print('first')
elif x == 2: print('second')
elif x == 3: print('third')
else: print('did not place')
```

This isn't exactly encouraged by PEP 8, but it's readable and not too unidiomatic.

Over the more than a decade since PEP 3103 was rejected, the issue of C-style case statements, or even the slightly more powerful version in Go, has been considered dead:

However, the idea of full ML-style pattern matching arises every few years, especially since languages like Swift and Rust have adopted it. The problem is that it's hard to get much use out of pattern matching without algebraic data types. While Guido has been sympathetic to the idea, nobody's come up with a proposal that fits into Python very well. (You can read my 2014 strawman for an example.) This could change with `dataclass` in 3.7 and some sporadic proposals for a more powerful `enum` to handle sum types, or with various proposals for different kinds of statement-local bindings (like PEP 3150, or the set of proposals currently being discussed on -ideas). But so far, it hasn't.

There are also occasionally proposals for Perl 6-style matching, which is basically a mishmash of everything from `elif` to regex to single-dispatch type-switching.

answered Apr 10 at 6:13

abarnert
220k 19 307 401

If you don't worry losing syntax highlight inside the case suites, you can do the

```
    2: """
print ('two')
""",
    3: """
print ('three')
""",
}.get(value, """
print ('None')
""")
```

Where `value` is the value.
In C, this would be:

```
switch (value) {
    case 1:
        printf("one");
        break;
    case 2:
        printf("two");
        break;
    case 3:
        printf("three");
        break;
    default:
        printf("None");
        break;
}
```

We can also create a helper function to do this:

```
def switch(value, cases, def
    exec cases.get(value, de
```

So we can use it like this for the example with one, two and three:

```
switch(value, {
    1: """
print ('one')
    """,
    2: """
print ('two')
    """,
    3: """
print ('three')
    """,
}, """
print ('None')
""")
```

answered May 25 '16 at 22:56

I was quite confused after reading the answer, but this cleared it all up:

```
def numbers_to_strings(argum
    switcher = {
        0: "zero",
        1: "one",
        2: "two",
    }
    return switcher.get(argu
```

This code is analogous to:

```
function(argument){
    switch(argument) {
        case 0:
            return "zero";
        case 1:
            return "one";
        case 2:
            return "two";
        default:
            return "nothing"
    }
}
```

Check the Source for more about dictionary mapping to functions.

edited Jun 19 '17 at 11:49

Nathanael Farley
224 2 15

answered Jun 14 '17 at 12:39

Yster
1,422 2 16 32

Inspired by this awesome answer. Requires no other code. Not tested. Realized that fall through does *not* work properly.

```
for case in [expression]:
    if case == 1:
```

```
        do_default()
```

edited Feb 15 at 13:17

answered Feb 5 at 1:54

Solomon Ucko
434 5 16

I would just use if/elif/else statements. I think that it's good enough to replace the switch statement.

answered Sep 15 '08 at 22:00

miya
739 1 8 18

Expanding on Greg Hewgill's answer - We can encapsulate the dictionary-solution using a decorator:

```
def case(callable):
    """switch-case decorator
    class case_class(object)
        def __init__(self, *
            self.args = args
            self.kwargs = kw

        def do_call(self):
            return callable(

return case_class

def switch(key, cases, defau
    """switch-statement"""
    ret = None
    try:
        ret = case[key].do_c
    except KeyError:
        if default:
            ret = default.do
```

```
@case
def case_1(arg1):
    print 'case_1: ', arg1

@case
def case_2(arg1, arg2):
    print 'case_2'
    return arg1, arg2

@case
def default_case(arg1, arg2,
    print 'default_case: ',

ret = switch(somearg, {
    1: case_1('somestring'),
    2: case_2(13, 42)
}, default_case(123, 'astrin

print ret
```

The good news are that this has already been done in NeoPySwitch-module. Simply install using pip:

```
pip install NeoPySwitch
```

edited May 23 '17 at 12:26

Community ♦
1 1

answered Apr 21 '17 at 7:17

Tom
161 12

I made this small and clean solution

```
result = {
    'case1':     foo1,
    'case2':     foo2,
    'case3':     foo3,
    'default':   default,
}.get(option)()
```

where foo1(), foo2(), foo3() and default() are functions

AlejandroQH
369 6 16

1 Wouldn't that execute all of the functions? A proper `switch` replacement would only run the option that was needed... – Brian Underwood Feb 20 at 16:20

Yes, for example if your variable option=="case2" your result=foo2() – AlejandroQH Feb 20 at 21:27

and so and so forth. – AlejandroQH Feb 20 at 21:41

Yes, I understand the purpose. But my concern is that if you only want `foo2()`, the `foo1()`, `foo3()`, and `default()` functions are all also going to run, meaning things could take a long time – Brian Underwood Feb 21 at 17:57

1 omit the () inside the dictionary. use `get(option)()`. problem solved. – timgeb Apr 10 at 6:00

1 2 next

protected by Jon Clements ♦ Aug 16 '15 at 10:58

Thank you for your interest in this question. Because it has attracted low-quality or spam

Would you like to answer one of these unanswered questions instead?