Thursday, March 28, 2019

Dan Bader 9:48 AM
@Dan Bader has joined the channel
@Dan Bader set the channel purpose: Let's chat about all things decorators

Chris Bailey 9:49 AM
@Chris Bailey has joined the channel

Geir Arne Hjelle 9:52 AM
@Geir Arne Hjelle has joined the channel

Jeremiah Cooper 9:55 AM
@Jeremiah Cooper has joined the channel

Kyle Stratis 9:55 AM
@Kyle Stratis has joined the channel

Laurent Stanevich 9:55 AM
@Laurent Stanevich has joined the channel

John Carey 9:55 AM
@John Carey has joined the channel

Roger Allen 9:55 AM
@Roger Allen has joined the channel

Aditya Mahapatra 9:56 AM
@Aditya Mahapatra has joined the channel

Thomas Davis 9:56 AM
@Thomas Davis has joined the channel

Noelle Anderson 9:56 AM
@Noelle Anderson has joined the channel

Rodrigo Vieira 9:56 AM
@Rodrigo Vieira has joined the channel

Hugh 9:56 AM
@Hugh has joined the channel

John Huff 9:56 AM
@John Huff has joined the channel

Matt Landers 9:56 AM
@Matt Landers has joined the channel

VINESH Chemmala PAUL 9:56 AM
@VINESH Chemmala PAUL has joined the channel

Dmytro Skorobohatow 9:56 AM
@Dmytro Skorobohatow has joined the channel

Dirk T. 9:56 AM
@Dirk T. has joined the channel

Jackie Wilson 9:56 AM
@Jackie Wilson has joined the channel

Jens Dittmar 9:57 AM
@Jens Dittmar has joined the channel

Dan Bader 10:00 AM
Hey and welcome everyone! So this is going to be our first ever Slack Q&A session, and today we're chatting about **Python decorators** :slightly_smiling_face: There's no real agenda for this session, so we're happy to go wherever you guys want to go and chat about all things related to decorators and their usage in Python.

Please try to use the Slack "threads" feature as much as possible so we can keep the chaos-level in the channel low :slightly_smiling_face: → https://medium.com/@mg/several-people-are-typing-using-slack-threads-by-default-to-reduce-distraction-and-chatter-7865918de02b

Geir Arne Hjelle 10:02 AM
Hey everybody! :rocket:

A question to kick things of: What's your favorite decorator?

> **Dan Bader** 10:03 AM
> Spontaneous answer: `functools.lru_cache`
> https://docs.python.org/3/library/functools.html#functools.lru_cache
> [+1] (2):
> **Geir Arne Hjelle**, **Ashish Cherian**
>
> **Roger Allen** 10:05 AM
> We just started using them, and the first one our group used is to help indent/outdent lines in an HTML file we were writing.
> [+1] (1):
> **Chris Bailey**
>
> **Mike Driscoll** 10:08 AM
> probably contextlib
>
> **Abhinav Ajitsaria** 10:08 AM
> The first one I used was a timing decorator I wrote to print the execution time of the decorated function. It's probably my most used decorator till date

[+1] (2):
Roger Allen, **Dan Bader**

Mike Driscoll 10:08 AM
we use a lot of logging decorators at my job, so that's up there as well
[+1] (2):
Chris Bailey, **Ashish Cherian**

Roger Allen 10:09 AM
@Mike Driscoll Do they work in conjunction with the logging library?

Dan Bader 10:14 AM
Actually, I think I wanna change my answer... `functools.wraps` is my favorite decorator :slightly_smiling_face:

Mike Driscoll 10:21 AM
I actually almost said `functools.wraps` too...
[grin] (1):
Dan Bader

@Roger Allen The logging decorators we do use Python's `logging` module. We also use an exception handling decorator too for catching unusual crashes
[+1] (1):
Abhinav Ajitsaria

Nick Singer 10:03 AM
Haven't written one yet. Looking here for inspiration!

Alex Riviere 10:03 AM
built in or home made?

Geir Arne Hjelle 10:03 AM
Either one!

Mahidhar 10:04 AM
can we pass lambda functions inside a decorator like this

```
def my_decorator(func):
    def wrapper():
        func()
    return  wrapper

@my_decorator(lambda x: x*2)
```

Geir Arne Hjelle 10:09 AM
Hi **@Mahidhar**
You can pass lambdas to deocrators, but not using the @ syntax. Instead you can wrap your lambda manually:

```
def my_decorator(func):
    def wrapper(args, *kwargs):
        func(args, *kwargs)
    return  wrapper

_ = my_decorator(lambda x: x**2)
```

Mahidhar 10:15 AM

what is the diffference between @ and _
I have seen this = *func() many times... i thought* means whatever is returned from the function...
What does it actually mean?

John Huff 10:05 AM

I'd suggest a decorator, decoratee, and decorated.

Dan Bader 10:09 AM

Ok so what's your favorite decoratee John? :smile:

Alex Riviere 10:06 AM

I'm partial to the unsync library
https://github.com/alex-sherman/unsync/

Dan Bader 10:07 AM

Yes! I love the concept
[heart] (1):
Alex Riviere

What have you used it for?
I thought it was so cool to just be able to say "I want to parallelize this bit of code here and it is cpu bound", and then let unsync work its magic

Alex Riviere 10:10 AM

Unfortunately our main codebase at work is still in 2.7 and i haven't been writing much python on my own, but i also work in JS all day and this makes it so much easier for me to wrap my head around parallelizing or asyncing stuff in python. it matches my mental model a whole lot more.
This is definitely one of those times where i don't feel like i need to go and learn how the internal mechanics of something works just to get up and running.

Dan Bader 10:56 AM

Yep, I thought it was a really smart design... didn't realize it was Py 2.x only, I'd love for it to thrive in Py 3-land as well :slightly_smiling_face:

Alex Riviere 11:36 AM

Yeah, i believe it is making use of asyncio, so 3.5+ only?

Pavel Krejsa 10:07 AM
If I start with decorators, first mentioned stuff is: Functions are first class citizens. What does that means? Is there something as second class citizen objects?
[+1] (1):
Mick Chanthaseth

Mike Driscoll 10:09 AM
As I recall, that means a function can be passed as a parameter to another function

Chris Bailey 10:10 AM
I go into depth in the Decorators 101 course, in brief you can use a function like other variables. Functions can be included in lists, dictionaries and be passed as arguments/parameters into other functions.

Mike Driscoll 10:10 AM
Looks like **@Dan Bader** covers it here as well - https://dbader.org/blog/python-first-class-functions
[+1] (1):
Chris Bailey

Python's Functions Are First-Class – dbader.org
Python's functions are first-class objects. You can assign them to variables, store them in data structures, pass them as arguments to other functions, and even return them as values from other functions.
[Image]

Chris Bailey 10:10 AM
https://realpython.com/lessons/functions-first-class-objects-python/

Functions as First-Class Objects in Python – Real Python
What does "Functions as First Class Objects" mean? You'll see in this lesson that it means that functions can be passed around into lists and used as arguments for other functions. In the next lesson you'll learn how to define functions inside functions.
[Image]

Pavel Krejsa 10:10 AM
O great, thanx a lot.

Mick Chanthaseth 10:12 AM
Could you explain second class citizens objects?

Chris Bailey 10:13 AM
In other languages functions are not allowed to be used in that way. In python everything is an object.

Nick Singer 10:14 AM
So it's safe to say that **all** objects are first-class--there are no second-class citizens?

Pavel Krejsa 10:15 AM
so everything in Python is object thus everything is first-class .. ?

Mike Driscoll 10:16 AM
I don't think Python has second class citizens. At least, I can't think of any examples

John Huff 10:18 AM
It might be worth noting that passing a function around supports the Open-Closed principle. (code should be open to extension while closed to modification or something like that.) Which means if my core code is something like def calc_something1(arg): return arg=arg+1 return calc_something=calc_something1return#note no parenthesis after something1 so it isn't invoked, it's just a reference to the functionreturnresult = calc_something(5)return # If I want to change what the variable calc_something does, I just set it to a different function reference.

Chris Bailey 10:19 AM
https://en.wikipedia.org/wiki/First-class_citizen

> **Wikipedia**
> **First-class citizen**
> In programming language design, a first-class citizen (also type, object, entity, or value) in a given programming language is an entity which supports all the operations generally available to other entities. These operations typically include being passed as an argument, returned from a function, modified, and assigned to a variable.

Mick Chanthaseth 10:26 AM
Ah. It's CS jargon. Good to know.

Dan Bader 10:12 AM
So I just did a quick grep through the realpython.com code base (a Django app) and I've defined zero custom decorators—however I use a bunch that are defined in Django and other libraries (e.g. `@cached_property` and whatnot).

Is this your experience too, that you mostly use pre-defined decorators coming from a library or framework?

Mike Driscoll 10:13 AM
We use mostly custom decorators at my current job, although we also use a few from Python's standard library

Laurent Stanevich 10:14 AM
I've only used a couple of custom utility decorators I wrote myself
Mainly some debug utilities, like a decorator that writes out enter/exit/timing details to a log/console

Pavel Krejsa 10:12 AM
Anyone ha OAuth2 decorators for Google API calls for Flask? I know Google released decorators for WebApp2 some time ago

> **Geir Arne Hjelle** 10:17 AM
> I haven't used oauth2 myself, but I see they have some convenience decorators: https://github.com/googleapis/oauth2client/blob/master/oauth2client/contrib/appengine.py#L470
>
> Has anybody else used them?
>
> **oauth2client/contrib/appengine.py:470**
> `
> class OAuth2Decorator(object):
> `
>
> <https://github.com/googleapis/oauth2client|googleapis/oauth2client>
>
> **Pavel Krejsa** 10:19 AM
> Note: oauth2client is now deprecated. No more features will be added to the libraries and the core team is turning down support. We recommend you use google-auth and oauthlib. For more details on the deprecation, see oauth2client deprecation.
>
> **Geir Arne Hjelle** 10:38 AM
> Ah, okay. Thanks for the heads up! :+1:
> [+1] (1):
> **Pavel Krejsa**

Nick Singer 10:14 AM
So it's safe to say that **all** objects are first-class--there are no second-class citizens?

Geir Arne Hjelle 10:17 AM
I haven't used oauth2 myself, but I see they have some convenience decorators: https://github.com/googleapis/oauth2client/blob/master/oauth2client/contrib/appengine.py#L470

Has anybody else used them?

> **oauth2client/contrib/appengine.py:470**
> `
> class OAuth2Decorator(object):
> `
>
> <https://github.com/googleapis/oauth2client|googleapis/oauth2client>

Mahidhar 10:17 AM
What is the significance of **args and *kwargs ... I can use it everytime I define a function... but that would be stupid without understanding it. When should we actually use them?

Laurent Stanevich 10:18 AM
Those refer to the unnamed and named parameters that got passed into the function

Paul Wildenhain 10:19 AM
`*args` allows you to pass an arbitrary amount of unamed parameters into a function.
so you could have `def sum_all_the_nums(**args):`
and then pass something like this through it
`

```
sum_all_the_nums(1, 2, 1)

  4

sum_all_the_nums(4, 5, 6)

  15
```

`

[+1] (1):
Ashish Cherian

Laurent Stanevich 10:19 AM
`args` is a tuple of all the unlabeled arguments, and `kwargs` is a dict of all the labeled arguments
[+1] (2):
Paul Wildenhain, **Ashish Cherian**

JL Diaz 10:21 AM
But the key is not the name of those parameters, which is irrelevant. It is the number of **` in front of them, and their order. The one with `*` will be a tuple with all unnamed parameters passed to the function, and the one with `*** will be a dictionary with all the named parameters passed to the function

Laurent Stanevich 10:21 AM
The * is a "splat" operator, which means that it explodes those collections into conventional data structures

Aditya Mahapatra 10:21 AM
Also, when you define a function with **args and ***kwargs, it let's you call the function without passing any args or kwargs if you don't want to

Geir Arne Hjelle 10:21 AM
Essentially **`args` will pick up all unnamed parameters sent in to a function and bind it to a tuple named `args` inside the function. Similarly `*`** `kwargs` will pick up all named parameters and bind it to a dictionary.

Actually the `` **and** `*` `` are the important part. You can name them what you like, althuogh `args` and `kwargs` are quite conventionaly. I often find it makes sense to give them more descriptive names though.

[+1] (3):

Chris Bailey, **Raghu**, **Mahidhar**

Dan Bader 10:21 AM

I like to think of them as a "placeholder" that says "here are all the arguments to that function, whatever they are" and if you use the placeholder you don't have to worry about function signatures changing -- e.g. your code can handle `func("a", 123, 42)` and `func("b", 222, 42, True, myarg=obj)` the same way

[+1] (1):

Mahidhar

John Huff 10:21 AM

This is very powerful. Googling packing and unpacking arguments should provide more info

Laurent Stanevich 10:22 AM

They are, indeed, very powerful -- but I have to admit that they're still a bane of my existence when it comes to auto-documentation

[+1] (3):

Geir Arne Hjelle, **JL Diaz**, **Kyle Stratis**

Geir Arne Hjelle 10:23 AM

One warning though: :blush: Don't run off and change all your signatures to `(**args, *kwargs)`, it will kill the readability of your code, as well as make the built-in documentation much harder to use ...

[100] (3):

Dan Bader, **Paul Wildenhain**, **Kyle Stratis**

Laurent Stanevich 10:27 AM

Yeah, that's the killer, for me -- more powerful "meta" functions (like decorators) that rely on generalized parameters become very difficult to document usefully, in a way that contributes to type-hinting, etc.

John Huff 10:32 AM

I've used the capability in my etl code and I believe they have significantly reduced the footprint/variation of what are similar but someways different etl streams. I pass in the extract, transform, and load functions from a code config file in which the signatures are explicit and there is also a get_arguments function which combines what I'm iterating over in my core code with the "common static arguments" known by the config file (e.g. read_csv vs read_fwf and other configuration arguments.) I believe this technique has been helpful in surfacing just about all configurations (e.g. #of header rows) in a declarative manner instead of burying them

in the midst of imperative code.

[+1] (4):

Chris Bailey, **Mahidhar**, **Kyle Stratis**, **Dan Bader**

Kyle Stratis 11:04 AM

@John Huff I'd love to see an example of this architecture if you can share

[+1] (1):

Laurent Stanevich

John Huff 11:40 AM

I'll ask at work. It's really generic so I don't see why not.

Kyle Stratis 4:55 PM

Looking forward to it!

Tony Jones 10:18 AM

Joined **#decorators-qa**

Mike Driscoll 10:18 AM

When you decorate a function, you have to account for its arguments and keyword arguments too

Roger Allen 10:21 AM

When do you define is an appropriate time to use a decorator?

Dan Bader 10:24 AM

Hah great question—I'd say any time where it makes your code easier to read and easier to maintain...more Pythonic. That's sort of a vague answer, but in my experience there's a fine line between using decorators for good vs making your code utterly confusing with them :slightly_smiling_face:

More an "art" rather than a "science", if that makes sense

But I'm curious to hear everyone else's perspective on this too!

Mike Driscoll 10:25 AM

There are times where you are not allowed to modify the code, but you can add a decorator to enhance said code. Or at least, that's one argument I've heard

Chris Bailey 10:25 AM

It can provide a layer of maintainability. Not having to repeat code.

Mike Driscoll 10:26 AM

I like the authentication decorators that `dj`ango uses

Dan Bader 10:26 AM

> Not having to repeat code.

:+1:

Geir Arne Hjelle 10:27 AM
I typically reach for decorators when I want to add some behavior that is not the main purpose of the function. There are a few examples in my article*, like caching a calculation, or just registering a function to a list of functions.

In one of my projects at work we used a `@register` decorator to dynamically create a list of models we could run, and which was then controlled by a configuration file.

* https://realpython.com/primer-on-python-decorators/
[+1] (3):
Abhinav Ajitsaria, **Laurent Stanevich**, **Jens Dittmar**

Chris Bailey 10:28 AM
I really like the registration system in the article!

Mike Driscoll 10:24 AM
I work with GUI interfaces a lot, so I will see things where we go into a screen, do something, and exit that screen. It I need to do that often, then it's almost always a case of needing a context manager and that brings you to using `contextlib` which has a decorator

Geir Arne Hjelle 10:29 AM
`@contextlib` is great for easily creating your own context manager! :smile:

Fahim Khan 11:13 AM
Where else we can use decorator apart from @contextlib in GUI program

Mike Driscoll 10:31 AM
For anyone wanting to see a really crazy decorator:
https://twitter.com/dabeaz/status/1064497836975362048
[joy] (2):
Geir Arne Hjelle, **Dan Bader**

Terry Spotts 10:33 AM
https://github.com/dabeaz/flail

dabeaz/flail
Ball and Chain Decorators
Stars
24
Language
Python
<https://github.com/dabeaz/flail|dabeaz/flail>|11/19/18, 7:40 AM

Mike Driscoll 10:34 AM

He's worth following on Twitter just for these random and interesting Python hacks

Dan Bader 10:35 AM
Btw I'm not sure if everybody has seen this, but the `Primer on Python Decorators` article by **@Geir Arne Hjelle** includes some bonus resources you may find helpful:

• Decorators cheat sheet: https://static.realpython.com/decorators-cheatsheet.pdf

• Decorators chapter from Python Tricks:
https://static.realpython.com/guides/the-power-of-python-decorators.pdf
[100] (4):
Abhinav Ajitsaria, **Paul Wildenhain**, **Shay Elmualem**, **Ilya S**
[+1] (7):
Doug Rohm, **Chris Bailey**, **Mirko Stojiljkovic**, **tchoumi kane charles wadel dylan**, **Raghu**, **Mahidhar**, **Peter Thomas**

Aditya Mahapatra 10:37 AM
ELI5: @property decorator?

Mike Driscoll 10:40 AM
It is the "pythonic" way of creating setters and getters in Python. There are lots of good tutorials on the subject online - https://www.programiz.com/python-programming/property
[+1] (1):
Aditya Mahapatra

Python @property: How to Use it and Why? - Programiz
You will learn about Python @property; pythonic way to use getters and setters.

https://www.python-course.eu/python3_properties.php
[+1] (1):
Aditya Mahapatra

Object oriented programming in Python: instance attributes vs. class attributes and their proper usage.

Dan Bader 10:41 AM
If you're wondering how `@property` is implemented internally, check out
https://stackoverflow.com/questions/17330160/how-does-the-property-decorator-work
[+1] (1):
Aditya Mahapatra

How does the @property decorator work?
I would like to understand how the built-in function property works. What confuses me is that property can also be used as a decorator, but it only takes arguments when used as a built-in

function ...

And this is super helpful too: https://docs.python.org/2/howto/descriptor.html#properties
[+1] (1):
Aditya Mahapatra

To see how property() is implemented in terms of the descriptor protocol, here is a pure Python equivalent:

Python 3 version: https://docs.python.org/3/howto/descriptor.html#properties

Aditya Mahapatra 10:42 AM
I do not know what descriptor protocol means :stuck_out_tongue:
But now that I have heard of it, I shall read of it! :slightly_smiling_face:

Mike Driscoll 10:43 AM
https://docs.python.org/3/howto/descriptor.html

Aditya Mahapatra 10:43 AM
Thank you both for the excellent resources!

Mike Driscoll 10:44 AM
Oops. I guess Dan already posted that one.
I wrote about descriptors on my blog once, but I don't think I did the subject justice

Aditya Mahapatra 10:44 AM
No worries! Can I have the link to your blog?

Mike Driscoll 10:45 AM
Sure. Here's the article too:
https://www.blog.pythonlibrary.org/2016/06/10/python-201-what-are-descriptors/
[+1] (1):
Aditya Mahapatra

Laurent Stanevich 10:40 AM
Is there much broader discussion around "decorators" in a functional programming context?
In the sense that there's potentially a very short line between "decorators" and "side effects"?

Mahidhar 10:44 AM
How do i use a function with decorator and not without decorator
@timer
def waste_some_time(num_of_times):

if case 1 run waste_some_time() with decorator
if case 2: run without decorator

Do I have to define the function twice to implement the above functionality?

Mike Driscoll 10:46 AM
You can't. Python doesn't support defining a function with the same time multiple times except with `@property`
You can kind of do function re-definitions using `functools.singledispatch`

Geir Arne Hjelle 10:47 AM
In this case I would first define the function:

```
def some_function():
    ...
```

Then you can decorate it, either each time you call it, or create a permanent decorated function:

```
if case == 1:
    my_decorator(some_function)()
elif case == 2:
    some_function()
```

or

```
decorated_func = my_decorator(some_function)

if case == 1:
    decorated_func()
elif case == 2:
    some_function()
```

[+1] (1):
Mahidhar

Laurent Stanevich 10:48 AM
I know this is just a hypothetical, but you'd also probably want to look at whether your `case` logic really just belongs in your decorator

Mike Driscoll 10:49 AM
@Geir Arne Hjelle Ooh...I had forgotten about that trick

Matt Landers 10:49 AM
you could potentially have the decorator take an arg (such as a debug bool) and use the same logic mentioned above within the decorator itself

Laurent Stanevich 10:50 AM
If you're choosing whether or not to apply the decorator based on some kind of "global" condition (like if a 'debug' flag is true), then you might as well just call the decorator all the time, and let it decide for itself whether or not to actually do anything
[+1] (4):
Mahidhar, **Roger Allen**, **Matt Landers**, **Blaise Pabon**

Geir Arne Hjelle 10:51 AM
Right, **@Matt Landers**, **@Laurent Stanevich**

I guess one example would be something like:

```
def debugger(func):
    @functools.wraps
    def wrapper_debug(args, debug=False, *kwargs):
        if debug:
            # do debug stuff
        value = func(args, *kwargs)
        if debug:
            # do more debug stuff
    return wrapper_debug
```

You can then use a debug=True extra argument to your decorated function to add debuggin

John Huff 10:51 AM
You don't have to use the @syntax. A decorator is just a function wrapping another function, so you could do case1 = decorator(waste_some_time)()returncase2=waste_some_time(). (Also, for understanding a decorator, I think using without the @syntax)
[+1] (1):
Laurent Stanevich

I should add that an important concept is that without the parenthesis, waste_some_time does not get invoked but is a variable reference to the function. Thus, in case one, you're passing in a variable function reference (the decoratee) to a function (the decorator) that returns another function (the decorated function). Now with the decorated function, we move onto the rightmost parenthesis above and invoke it.
[heart_eyes] (1):
Blaise Pabon

Chris Bailey 10:47 AM
If there was a Decorators 201 video course, what would you want it to cover?

Aditya Mahapatra 10:56 AM

A section on the most commonly used built-in decorators and how they function
[simple_smile] (1):
Chris Bailey
[+1] (1):
Dirk T.

Dirk T. 11:59 AM
More examples
use of decorators in django

Mahidhar 10:49 AM
In all the videos you have be saying 'rep'. what is the difference between 'repr' and terminal or console?

Chris Bailey 10:51 AM
I am referring to REPL (Read, Eval, Print and Loop). It is accessed from a terminal, but is a live python coding environment. You can access it by typing Ptyhon or Python3.
[+1] (1):
Mahidhar

Dan Bader 10:51 AM
A couple of terms here:

"repr" -> usually refers to `__repr__` and `repr()` which are used for to-string conversion of Python objects

"REPL" -> "Read Eval Print Loop" == an interactive (Python) interpreter session
For the `__repr__` / `repr()` stuff, check out this course:
https://realpython.com/courses/pythonic-oop-string-conversion-__repr__-vs-__str__/
:slightly_smiling_face:

Pythonic OOP String Conversion: __repr__ vs __str__ – Real Python
In this tutorial series you'll do a deep dive on how Python's to-string conversion using the __repr__ and __str__ "magic methods" works and how you can add implement them in your own classes and objects.
[Image]

Chris Bailey 10:52 AM
In my videos I'm using a REPL called bpython. Than adds code hinting etc.
[+1] (1):
Mahidhar

https://bpython-interpreter.org

Dan Bader 10:52 AM

bpython is the best!
REPL: https://en.wikipedia.org/wiki/Read%E2%80%93eval%E2%80%93print_loop

> **Wikipedia**
> **Read–eval–print loop**
> A read–eval–print loop (REPL), also termed an interactive toplevel or language shell, is a simple, interactive computer programming environment that takes single user inputs (i.e., single expressions), evaluates them, and returns the result to the user; a program written in a REPL environment is executed piecewise. The term is usually used to refer to programming interfaces similar to the classic Lisp machine interactive environment. Common examples include command line shells and similar environments for programming languages, and is very characteristic of scripting languages.

Mahidhar 11:00 AM
print(f"{func._*name*_!r}"
r in the above expresion referes to REPL or *repr*

Geir Arne Hjelle 11:00 AM
`repr`

without the `!r`, the `f-string uses the `str` representation by default. For `strings, the difference is that `!r` adds quotes around your string
[+1] (1):
Mahidhar

Mahidhar 11:03 AM
is this r and the r used in regular expressions the same? like re.sub(r"\s+", "", "hi man")

Dan Bader 11:04 AM
Ha great question! The `r` at the start of a string (as used for regexes) denotes a "raw string"
Raw strings have different escaping rules, like you don't need to escape backslashes
Trying to find a good resource to share here

Geir Arne Hjelle 11:05 AM
Right, no, those are two separate `r`-concepts
[+1] (1):
Mahidhar

Dan Bader 11:06 AM

> Both string and bytes literals may optionally be prefixed with a letter 'r' or 'R'; such strings are called raw strings and treat backslashes as literal characters. As a result, in string literals, '\U' and '\u' escapes in raw strings are not treated specially. Given that Python 2.x's raw unicode literals behave differently than Python 3.x's the 'ur' syntax is not supported.

https://docs.python.org/3/reference/lexical_analysis.html
[+1] (1):
Mahidhar

Mahidhar 11:06 AM
Now I get why the community is called "Real Python". Till now I think I have been doing dummy python from stackoverflow
[+1] (1):
Chris Bailey
[rolling_on_the_floor_laughing] (2):
Dan Bader, **Geir Arne Hjelle**
[realpython] (2):
Dan Bader, **Mahidhar**

:realpython: ... oh so there a emoticon also... great
[realpython] (1):
Geir Arne Hjelle

Augusto Valdivia 10:49 AM
@Chris Bailey Thank you for the `Decorator` tutorial. I am doing right now and it is helping me to growth my `Python` skilss
[grinning] (1):
Chris Bailey
[partyparrot] (1):
Dan Bader
[+1] (1):
Mahidhar

Chris Bailey 10:53 AM
Thank you!

Jens Dittmar 10:58 AM
Indeed, it's been a really good one :blush:

Aditya Mahapatra 10:51 AM
How and why was the @ symbol chosen to denote a decorator?

Mike Driscoll 10:52 AM
https://www.python.org/dev/peps/pep-0318/#why
[+1] (3):
Chris Bailey, **Aditya Mahapatra**, **Mirko Stojiljkovic**

PEP 318 -- Decorators for Functions and Methods
The official home of the Python Programming Language

Aditya Mahapatra 10:54 AM
Excellent! Is there a list of the symbols being used in python? @ for decorator, # for comments and so on?

Geir Arne Hjelle 10:55 AM
From the same PEP:

> Barry Warsaw named this the 'pie-decorator' syntax, in honor of the Pie-thon Parrot shootout which occurred around the same time as the decorator syntax, and because the @ looks a little like a pie. (https://www.python.org/dev/peps/pep-0318/#background)

[+1] (1):
Aditya Mahapatra

PEP 318 -- Decorators for Functions and Methods
The official home of the Python Programming Language

Mike Driscoll 10:57 AM
Maybe this is what you're looking for: https://docs.python.org/3/genindex-Symbols.html
[+1] (1):
Aditya Mahapatra
[open_mouth] (2):
Mirko Stojiljkovic, **Chris Bailey**

Aditya Mahapatra 10:59 AM
Perfect!

Dan Bader 11:01 AM
Hah, didn't know about the parrot shootout! That's amazing :slightly_smiling_face: :shuffle_parrot:

John Huff 10:59 AM
One dimension I think of when using decorators is whether I'm adding general functionality to a specific function (and will use the @shortcut syntax) versus adding specific functionality to the "workflow" without polluting the workflow code. (e.g. adding specific tests/validation/reconciliations to the workflow code.
[+1] (1):
Mirko Stojiljkovic

Aditya Mahapatra 11:09 AM
Has anyone ever been asked a decorator related question in an interview?

Mirko Stojiljkovic 11:10 AM
Does that include `@property`?

Mike Driscoll 11:10 AM
Yes

John Huff 11:11 AM
Yes, just last month "What is a decorator" The interviewer was happy with my answer, but upon reflection, I wished I had used the terms "decorator function", "decoratee function", and "decorated function".

Mike Driscoll 11:11 AM
I have never seen the term "decoratee function" before

Mahidhar 11:12 AM
Not in interview...but in my previous workspace there was a guy who was using few wrappers and decorators for timer functions and database updation functions

Aditya Mahapatra 11:12 AM
What was your answer, if you don't mind sharing? And what level position was this interview for?
@John Huff

John Huff 11:14 AM
decorated_func = decorator_func(decoratee_func) It's my own terminology that came to me when thinking how to be more articulate and concise when put on the spot in a interview.

Mirko Stojiljkovic 11:17 AM
@John Huff :wave: That's a cool term!

John Huff 11:22 AM
I began my answer with stating simply, it's a function that wraps another function. I then explained how without parenthesis a function doesn't get invoked. I then wanted to get in packing and unpacking of keyword arguments, and started down that path with too many words, and then apologized to the interviewer saying let me get back to your explicit question. The interviewer then said, "You know what a decorator is, let's move on". The interview was for a senior production back-end quant web service developer. I have light web service experience (I read up frantically on the XmlWebRequest standard) and came in 3rd out of 6 based mostly on my general Python/Pandas knowledge in spite of my lack of back-end web service experience.
[+1] (2):
Aditya Mahapatra, **Jens Dittmar**

Aditya Mahapatra 11:28 AM
That's great **@John Huff**. Thanks for the answer.

Dan Bader 11:19 AM
Wow—that hour just flew by :slightly_smiling_face: That was fun! Thanks for coming to our first ever Slack Q&A everybody!

I'll leave this channel open & active and I'll also hang around here for a while longer, so feel free to keep going!

[+1] (4):
Mirko Stojiljkovic, **Mahidhar**, **Vincenzo Fiorentini**, **Laurent Stanevich**
[fast_parrot] (1):
Aditya Mahapatra

Mahidhar 11:21 AM
The videos are great... i was facing a problem with api requests... but the slow_down function taught in section 3 solved my problem.

[grinning] (2):
Chris Bailey, **Dan Bader**

Decorators are awesome... but can we have some short project ideas or exercises or simple sideas to try our understanding on decorators and hence improve the skill?

Dan Bader 11:23 AM
I'd love to do a decorators quiz at some point—I think we could have some fun exercises and questions in that one :slightly_smiling_face:

[realpython] (1):
Mahidhar
[+1] (1):
Sean Yang

Mirko Stojiljkovic 11:25 AM
That's a good idea.

Mirko Stojiljkovic 11:25 AM
@Mahidhar I've first made a decorator that makes the return value of a function always positive. Later, I've created a decorator that opens and closes the connection to the database, while the function itself handles DB stuff. I did that just for fun.

Augusto Valdivia 11:25 AM
@Dan Bader can `decorator's` be used in a `unit test`?

Mike Driscoll 11:32 AM
I believe you can, but you have to be careful to use `functools.wraps` on those decorators as some testing libraries don't like it when the function signature changes

Augusto Valdivia 11:34 AM
Ok - Thank you

Vincenzo Fiorentini 11:25 AM
I could join only now and I am about to take a plane :face_with_raised_eyebrow: : so will this chat remain available in the future? Thx !

Aditya Mahapatra 11:27 AM
It will. And Dan's gonna keep it open so you can ask questions later as well :slightly_smiling_face:
[+1] (1):
Vincenzo Fiorentini

Dan Bader 11:30 AM
Yep! At some point Slack will delete older messages, but I'll try to find a way to export the chat log and add it as a download on the website. It'll definitely still be around after your trip **@Vincenzo Fiorentini** :slightly_smiling_face:

Vincenzo Fiorentini 11:35 AM
Great ! Thx !

Vincenzo Fiorentini 11:26 AM
Btw I enjoyed immensely the decorator video course !
[grinning] (1):
Chris Bailey

Mahidhar 11:30 AM
Can we use decorators in for loop ...
Like in the PLUGINS = dict() example of video

@register func1()
@register func2()

Can I run a for loop such that
[@registerx for x in [func1, func2, func3, func4]]

Dan Bader 11:31 AM
You should be able to do this:

```
[register(x) for x in [...]]
```

(No @ prefix)
[realpython] (1):
Mahidhar

Decorators are just functions at the end of the day
[+1] (1):
Mahidhar

Laurent Stanevich 11:33 AM
Yeah, this kind of goes back to the whole "first-class" idea that came up earlier -- in some ways,

pretty much *everything* in Python is a function
"Decorators" are really just a syntactic shortcut that makes it easier to use them in one specific way
[+1] (1):
Mahidhar

(Namely, when you have a function that acts on *other* functions)
It's just one more example of where there are 5-10 different ways of doing the same thing in Python :wink:
[smile] (1):
Dan Bader

Mirko Stojiljkovic 11:35 AM
Anyone used **class decorators** recently?

Mike Driscoll 11:44 AM
Only in tutorials
[grin] (2):
Mirko Stojiljkovic, **Rohit Kumar**
[+1] (1):
Mirko Stojiljkovic

Laurent Stanevich 11:51 AM
Oh, and I forgot about what's maybe the most useful "standard" decorator: `@dataclass`
[+1] (1):
Mirko Stojiljkovic

Dan Bader 11:52 AM
Yes that's a great one!
Throwing this in here for reference: https://realpython.com/python-data-classes/ (by **@Geir Arne Hjelle**)
[+1] (2):
Mirko Stojiljkovic, **Laurent Stanevich**

The Ultimate Guide to Data Classes in Python 3.7 – Real Python
Data classes are one of the new features of Python 3.7. With data classes you do not have to write boilerplate code to get proper initialization, representation and comparisons for your objects.
[Image]

Mike Driscoll 11:54 AM
ooh. I haven't used that one yet

Geir Arne Hjelle 11:57 AM

Hehe, yes, I should have mentioned that one :blush:

Laurent Stanevich 12:02 PM
`@dataclass` *rocks*. I've always preferred to wrap classes around my key data structures (in any language), but that often implies a lot of extra time and effort creating and maintaining all the various getters/setters, etc.
`@dataclass` makes it much, much easier to "do things right", without all the added dev work

Mike Driscoll 12:18 PM
I still don't get to do production work in Python 3, so I won't be having this opportunity for a while

Sunday, May 12, 2019

Mahidhar 8:45 PM
A great talk by Geir Arne Hjelle. Got another project application of generators. But here are few questions.
Q1) Why do we need setuptools or pyplugs. I have heard that these days containerizing is more popular. Can we not not containerize (docker) - readers, models, filters, writers, notifiers and components. I am new to these topics but I see some parallels or intersection. What is the relaton between Dockers and pyplugs. If there is no relation how to take advantage of both for better project management.
Q2) What are Notifiers? Are these Android or iOs App notifications? If yes, How can we code it in python and integrate them into respective platforms?
Q3) Can the Components section have non-python elements, like javascript tools such as React Components or React-Native components or widgets etc.? If yes, can we integrate a React component to a python function using the same logic as used for _REGISTERS in this video? If possible please illustrate?

https://www.youtube.com/watch?v=98s9YfoXB68&__s=5k9owshowq38h6egis7s

PyCon 2019
Geir Arne Hjelle - Plugins: Adding Flexibility to Your Apps - PyCon 2019

Monday, May 13, 2019

Geir Arne Hjelle 5:56 AM
Hi **@Mahidhar**, and thanks for the shout-out. Glad you liked the presentation.

1) Docker vs setuptools vs pyplugs: These all solve a similar problem as you point out, but they work at different levels. A (Docker) container runs a full process, in our case often a Python program. In other words, the smallest component is a program/process. Pyplugs, on the other

hand, modularizes the python code itself. A pyplugs-plugin is usually a function or method. I don't think it makes too much sense to use pyplugs at the program/process level, and similarly using Docker at the function level is maybe possible, but seems like it will be complicated and have too much overhead. I have happily used both Docker and Pyplugs together: Create a plugin-based program and stick it in a Docker container. Regarding setuptools: I have only used it to install Python packages and create entrypoints that work as command-line programs. At the moment, setuptools is used to install Python packages and is such an important part of the Python ecosystem. As was pointed out in one of the questions after my talk, it can also be used to create and discover some kind of plugins. I don't really have any experience with this use.
2) I mentioned notifiers very briefly as an example of a plugins-package. I have not done any mobile development myself. In my case, I had a long-running program, and created different notifiers like "email_sender", "write_to_log", and so on. With these things coded up as plugins, it was easy to add new notifiers, and choose which ones I wanted to "subscribe" to (i.e. have the program call)
3) The pyplugs gives you a more flexible way of calling Python functions. These are just normal Python functions and can then do whatever you can usually do with Python functions. I have used plugins to build a GUI out of separate components using the `tkinter` framework. Essentially, I had a class that implemented the GUI. Different components (like tabs, dropdowns, buttons etc) where implemented in their own classes, and each such component class was registered. I then had a simple configuration file where I could say things like `tabs = run, analysis, graph` and the program would add those tabs to the GUI. I could then change around the GUI simply by editing the configuration file. I don't have much experience with javascript or setting up javascript from Python, but if you already have functionality for doing that, then it shouldn't be hard to integrate pyplugs into that.

Mahidhar 7:45 AM
Thanks for your answer. It clarified many doubts. I am happy to be RealPython member and I am learning a lot from RealPython videos and slack community. If possible, in future Can we have a short tutorial or video on "How to create a plugin-based program (or any general python based program) and stick it in a Docker container".

Geir Arne Hjelle 2:54 PM
(**@Dan Bader**) In a sense this sounds like two different tutorials: one on how to structure Python programs with plugins, and one on how to put python programs into a docker container. I think both are good ideas. We'll definitely add them to our list of tutorial ideas :blush:

Do also check out the **#tutorial-ideas** channel :grinning:
[partyparrot] (2):
Dan Bader, **Mahidhar**