Algorithms

- We have four classes on algorithms
- We'll need to create our own functions in order to encode algorithms
- We'll start with review of functions

Back <<< to the future

- A function takes in 0 or more arguments and returns 0 or 1 values.

```python
def my_function(str1, str2):
  return "The concatenation of str1 and str2 is " + str1 + str2
```

- This function takes two arguments, `str1` and `str2`. It returns another string.
- `def` *defines* the function. It doesn't run what's inside the function. In order to use the function, we have to *call it*, for example:

```python
result = my_function("hello ", "world")
```

Functions without arguments

- We can have a function without arguments that returns a value:

```
def the_answer_to_life_the_universe_and_everything():
  return 42
```

Function with arguments, with no returns

- We can have a function with arguments that doesn't return a value

```
def draw_regular_polygon(nsides, r):
  # Draws a polygon with lines.
  for i in range(nsides + 1):
    line(r * cos(i / float(nsides) * 2 * PI),
         r * sin(i / float(nsides) * 2 * PI),
         r * cos((i + 1) / float(nsides) * 2 * PI),
         r * sin((i + 1) / float(nsides) * 2 * PI))
```

Function without arguments and no return

- We can have a function without arguments and no return value

```
def setup():
  pass
```

Why choose one type of function over another?

- Depends on what you want to do
- Functions with no returns are used for their side effects (drawing, printing, logging, writing to a file, etc.)
- Functions with arguments and returns and no side effects are a lot like what mathematicians call functions. We can use them to implement mathematical functions, algorithms, etc.
- My advice: try to avoid creating functions that has both a return and a side effect.

How do you name things?

- You can name a function whatever you want (alphanumeric, underscore and digits).
- Your arguments can be named whatever you
- In Python, people like to name functions and arguments to start with a lowercase letter and words to be separated with underscores, e.g. `my_very_long_function`
- If you use a meaningful name, your code will be a lot easier for others (and future you) to read.

How long should a function be?

- Nothing stops you from making an if/else with 300 cases (google `VVVVVV source code`), but generally it's not a great idea
- If your function doesn't fit on one screen, that makes it harder to read and reason about.
- If you use more than 3-4 arguments, it becomes hard to figure out which argument does what.
- Big functions are hard to test - your function should ideally do *one thing*.

Scope and execution

- Functions have their own scope: they have their own variables
- When you call a function from another function, you start "from scratch": you just get the variables that come from the arguments of the function.

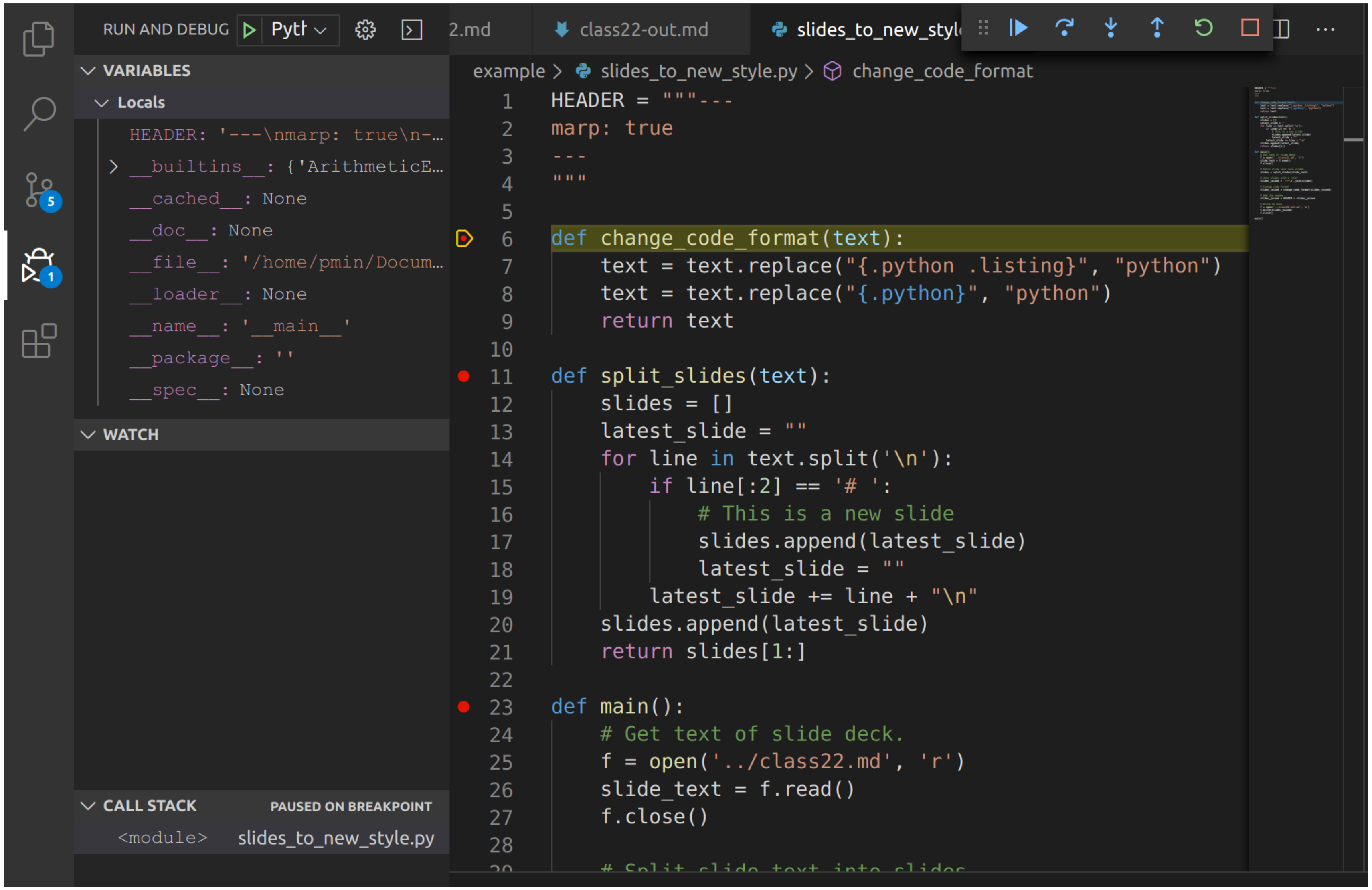
```python
d = 40

def fun_called(b, c):
  print(a)

def fun_caller():
  a = 10
  fun_called(20, 30)  # Error!
```

- In python, you also get read-only access to the module-level variables within functions
- To be able to read and assign, you have to use `global`

Illustration with a debugger



```
HEADER: '---\nmarp: true\n-…
> __builtins__: {'ArithmeticE…
__cached__: None
__doc__: None
__file__: '/home/pmin/Docum…
__loader__: None
__name__: '__main__'
__package__: ''
__spec__: None
```

```python
HEADER = """---
marp: true
---
"""

def change_code_format(text):
    text = text.replace("{.python .listing}", "python")
    text = text.replace("{.python}", "python")
    return text

def split_slides(text):
    slides = []
    latest_slide = ""
    for line in text.split('\n'):
        if line[:2] == '# ':
            # This is a new slide
            slides.append(latest_slide)
            latest_slide = ""
        latest_slide += line + "\n"
    slides.append(latest_slide)
    return slides[1:]

def main():
    # Get text of slide deck.
    f = open('../class22.md', 'r')
    slide_text = f.read()
    f.close()
```

Functions or classes?

- Python likes functions
- C# likes classes

Let's practice this

- Four functions:

```
def get_candle_icon():
  # Takes in no arguments, returns a string character for the candle (e.g. i)
  print ""

def prepare_cake(nstages, ncandles):
  # Takes in two ints, nstages and ncandles, returns a string cake of the right shape
  return ""

def print_cake(cake):
  # Prints the string cake to the console, returns nothing.
  pass

def print_happy_birthday():
  # Prints happy birthday to the console
  pass

print_happy_birthday()
print_cake(prepare_cake(3, 35))
```

Expected output

```
 iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
-------------------------------------
|                                   |
-------------------------------------
|                                   |
-------------------------------------
|                                   |
-------------------------------------
```

Algorithms

- Recipe for accomplishing a computational task
- The earliest algorithms predate computers

Muhammad ibn Musa al-Khwarizmi

- Persian mathematician (780-850) who first demonstrated quadratic equations
- In the 12th centery, his book on numerals introduced the Western world to Hindu-Arabic numbers.

Checking whether an integer is prime

- A number `N` is prime if the only integers that divide `N` are 1 and `N`.
- To check whether `N` is prime, we can check each number individually from 2 to sqrt(N) to see whether they divide `N`.

Checking the intuition behind the algorithm

Let's try the number 53.

Writing some pseudocode

```
function is_prime(N : integer) -> Boolean
  for x from 2..floor(sqrt(N))
    if x divides N then return True
  return False
```

The sieve of Eratosthenes

- One of the most famous ancient algorithms
- Algorithm to find prime numbers
- https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes

Coding up the sieve