Computational Structures in Data Science

UC Berkeley EECS
Lecturer
Michael Ball

Lecture #3: Loops and Functions

January 31, 2020

https://cs88.org

Administrivia

- More spots opened for lab sections
- Please try to attend labs you signed up for. (See Piazza)
- Reminder: iClickers next week.
  - Can register them at any time during the semester.
- We're going to be doing live coding, so review videos, not just slides.

Computational Concepts Today

- Conditional Statement
- Functions
- Iteration

Things you can do now:

- Write a program that makes a decision.
- Write your own functions
- Use loops so you can process lots of data.

A Brief Review: Files, Terminals

- This is mostly lab 0 review.
- It will take time to get used to everything!
- Things we'll do:
  - Use the command line to run files
  - Review the difference between notebooks and files

Let's talk About Python

- **Expression** `3.1 * 2.6`
- **Call expression** `max(0, x)`
- **Variables**
- **Assignment Statement** `x = <expression>`
- **Define Function:** `def` <function name> `(`<parameter list>`):`
- **Control Statements:** if …
  for …
  while …
  list comprehension

Conditional statement

- Do some statements, conditional on a *predicate* expression

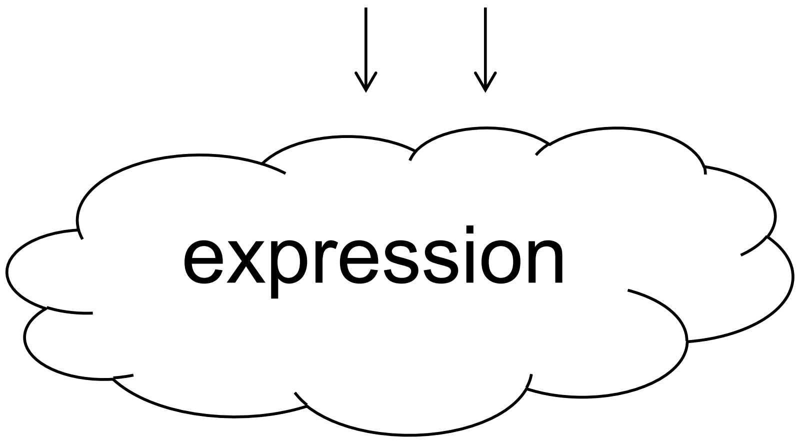
```
if <predicate>:
       <true statements>
else:
       <false statements>
```

- Example:

```
if (temperature>98.6):
       print("fever!")
else:
       print("no fever")
```

Defining Functions

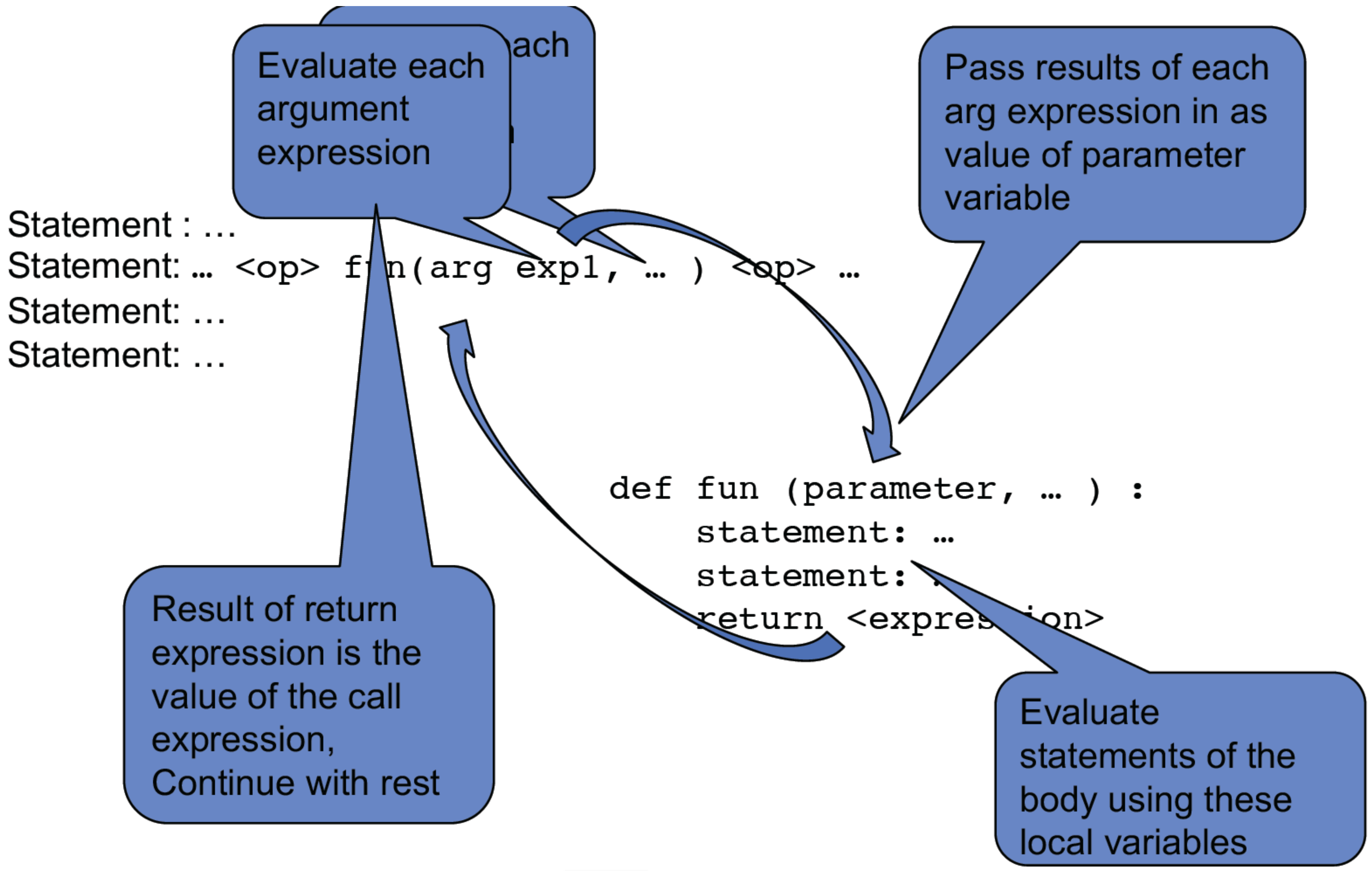
```
def <function name> (<argument list>) :
```



```
return
```

- **Abstracts an expression or set of statements to apply to lots of instances of the problem**
- **A function should *do one thing well***

Functions: Calling and Returning Results



Evaluate each argument expression

ach

Pass results of each arg expression in as value of parameter variable

```
Statement : …
Statement: … <op> fun(arg exp1, … ) <op> …
Statement: …
Statement: …
```

```
def fun (parameter, … ) :
    statement: …
    statement: …
    return <expression>
```

Result of return expression is the value of the call expression, Continue with rest

Evaluate statements of the body using these local variables

Functions and Arguments

```
>>> x = 3
>>> y = 4 + max(17, x + 4) * 0.5
>>> z = x + y
>>> print(z)
15.5
```

```
def max(x, y):
    return x if x > y else y
```

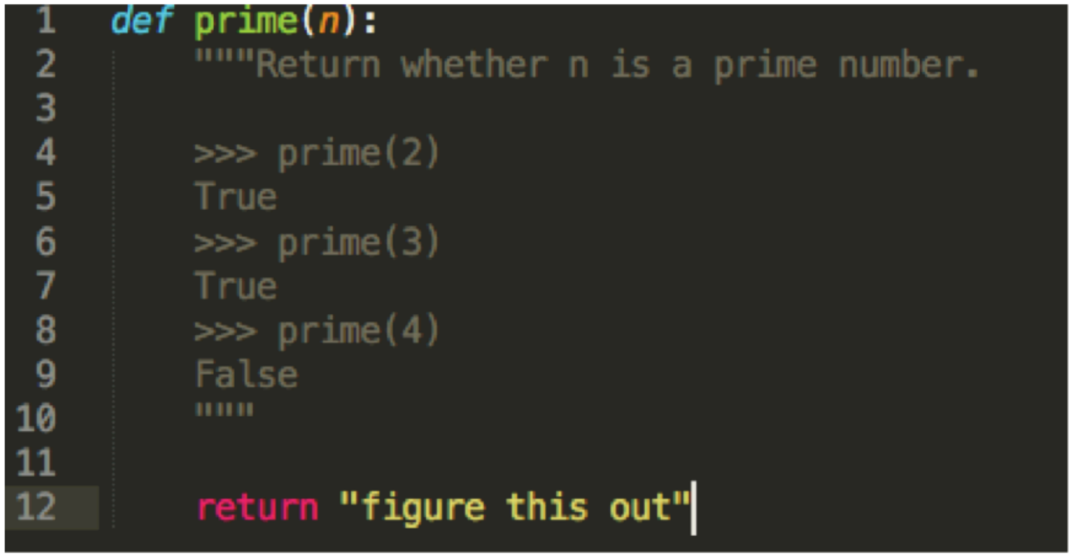
```
def max(x, y):
    if x > y:
        return x
    else:
        return y
```

How to write a good Function

- **Give a descriptive name**
  - Function names should be lowercase. If necessary, separate words by underscores to improve readability. Names are extremely suggestive!
- **Chose meaningful parameter names**
  - Again, names are extremely suggestive.
- **Write the docstring to explain *what* it does**
  - What does the function return? What are corner cases for parameters?
- **Write doctest to show what it should do**
  - **Before** you write the implementation.

Python Style Guide: https://www.python.org/dev/peps/pep-0008/

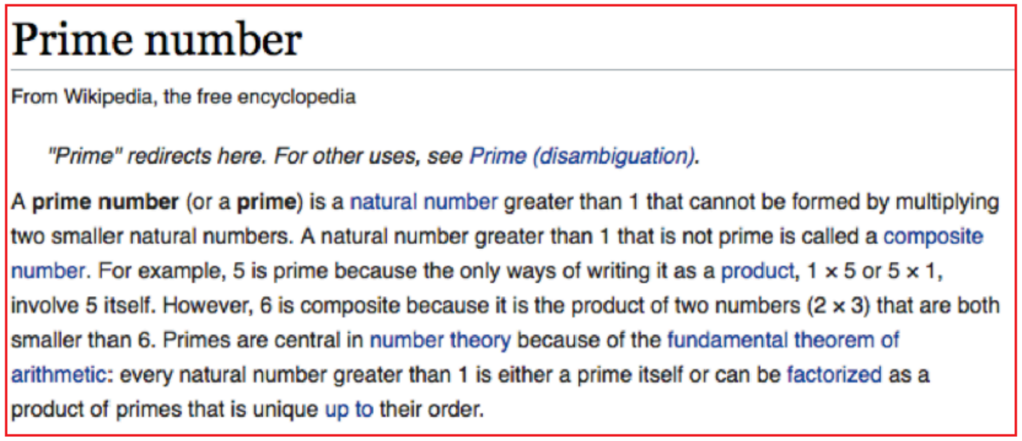
Example: Prime Numbers



```python
def prime(n):
    """Return whether n is a prime number.

    >>> prime(2)
    True
    >>> prime(3)
    True
    >>> prime(4)
    False
    """

    return "figure this out"
```



Prime number

From Wikipedia, the free encyclopedia

"Prime" redirects here. For other uses, see Prime (disambiguation).

A **prime number** (or a **prime**) is a natural number greater than 1 that cannot be formed by multiplying two smaller natural numbers. A natural number greater than 1 that is not prime is called a composite number. For example, 5 is prime because the only ways of writing it as a product, 1 × 5 or 5 × 1, involve 5 itself. However, 6 is composite because it is the product of two numbers (2 × 3) that are both smaller than 6. Primes are central in number theory because of the fundamental theorem of arithmetic: every natural number greater than 1 is either a prime itself or can be factorized as a product of primes that is unique up to their order.

Why do we have prime numbers?

https://www.youtube.com/watch?v=e4kevnq2vPI&t=72s&index=6&list=PL17CtGMLr0Xz3vNK31TG7mJlzmF78vsFO

`for` statement – iteration control

- **Repeat a block of statements for a structured sequence of variable bindings**

```
<initialization statements>
for <variables> in <sequence expression>:
    <body statements>

<rest of the program>
```

```
def cum_OR(lst):
    """Return cumulative OR of entries in lst.
    >>> cum_OR([True, False])
    True
    >>> cum_OR([False, False])
    False
    """
    co = False
    for item in lst:
            co = co or item
    return co
```