Computational Structures in Data Science

Lecture 3:
Functions and Loops

Berkeley
UNIVERSITY OF CALIFORNIA

Announcements (June 20 2024)

- Upcoming due dates (11:59 pm PST)
  - Lecture 01 Self Check: June 19 (can still submit late!)
  - Lecture 02 Self Check: June 20 Lab 00: June 22
- HW 01 released today! (due: June 24)
- Lab01 released today! (due: June 24)
- Discussion 01 released today! (not graded)
- Tip: each week will generally have two labs and two homeworks.

Announcements (June 20 2024)

- ALL office hours starts today (Thursday, June 20th)!
  - Reminder: Office hour Zoom links can be found:
    - Ed "Course Index" post, Course staff page, bCourse Zoom page
    - Instructor, TA's, and Tutors hold office hours each week.

Computational Structures in Data Science

Learning Process & Debugging

Berkeley
UNIVERSITY OF CALIFORNIA

Process NOT Memorization

- This is not a class about memorization.

- This is a class about *problem solving* and *process.*

- You will not know everything, but you will be able to figure it out.

- Focus on building intuition!

  - **Predict** what will happen **first**

  - Then **try and inspect**

  - Now, Figure out **why**!

  - Was your prediction correct or incorrect?

Computational Structures in Data Science

Python: Definition

Berkeley
UNIVERSITY OF CALIFORNIA

Learning Objectives

- Create your own functions.

- Write a loop to run the same code multiple times

- Use conditionals to control when a loop stops

Let's talk Python

- Expression   `3.1 * 2.6`

- *Call* expression   `max(0, x)`

- Variables   `my_name`

- Assignment Statement   `my_name = <expression>`

- Define Statement: `def function_name(<arguments>):`

- Control Statements: `if …`
                               `for …`
                               `while …`

- Comments   `# Text after the # is ignored.`

Variables In Python

- Variables "bind" (or assign) a name to a value (or expression)
- Variables can also come from function arguments
- Python has some specific rules about names...
  - Don't memorize them all!
  - Mostly: **No spaces**, use _
- Important: Use meaningful names!
  - It's a bit embarrassing to come to OH and try to explain the purpose of "butt" ☺ (This actually happened!)
- `my_favorite_class = 'C88C'`

Functions in Python

- We "define" them with def
- We typically name_them_using_underscores ("Snake case")
- The first line ends in a :
- The body is indented by 4 spaces
- Arguments (parameters) create 'names' that exist only in our function
- Most functions will return a value, but some do not.

```python
def print_greet(name):
    print("Hello, " + name)
def greet(name):
    return "Hello, " + name
```

Aside: String and Text

- Strings, or sequences of text are incredibly common!

- In Python we use ' or "

- We combine strings with +, or by using *string interpolation:*

- f-strings allow us to embed an expression inside some text!

```
def print_greet(name):
    # print("Hello, " + name)
    print(f"Hello, {name}")
```

Defining Functions

- Abstracts an expression or set of statements to apply to lots of instances of the problem

- A function should do one thing well

- arguments become accessible inside the function body.
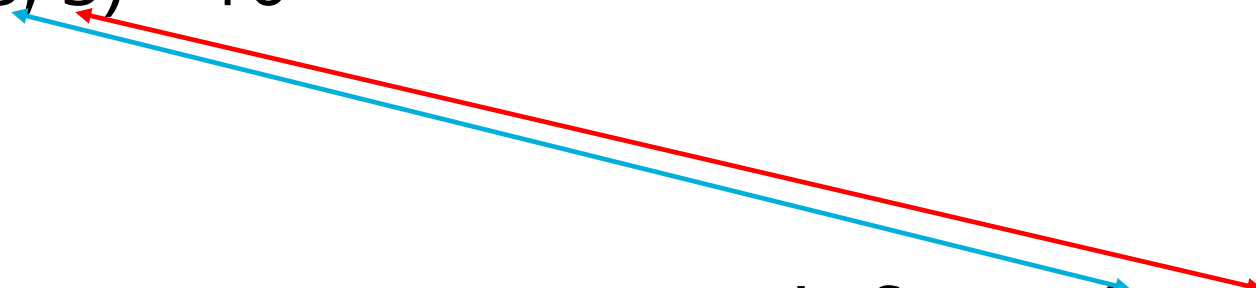
def <function name> (<argument list>) :

↓ ↓
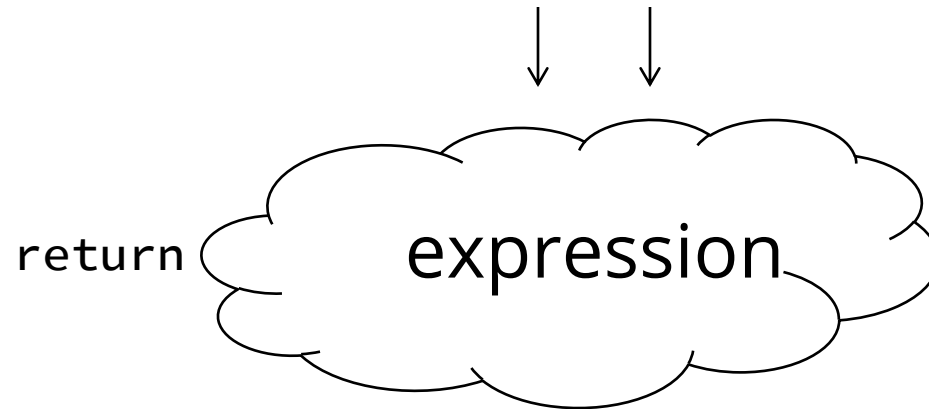
return expression

Functions: Example

- \>>> y = 5

- \>>> x = 3

- \>>> z = max(3, 5) * 10

- \>>> z

- 50

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

Returns and Values

- All functions always return SOME value.

- If you don't specify return, the value is None.

- Using print does not change how the function works, but does affect the output **displayed to the human user**.

Returns and Values: print()

- **Very common misconception**: do NOT use print() to return a value in a function (it returns None)

- Example:

```
def triple_it_v1(some_num):
    return some_num * 3


def triple_it_v2(some_num):
    print(some_num * 3)
```

```
# what do these output in the
interpreter?
>>> val1 = triple_it_v1(2)
>>> val1 + 1
...?
>>> val2 = triple_it_v2(2)
>>> val2 + 1
...?
```

Returns and Values: print()

- **Very common misconception**: do NOT use print() to return a value in a function (it returns None)

- Example:

```
def triple_it_v1(some_num):
    return some_num * 3

def triple_it_v2(some_num):
    print(some_num * 3)
```

```
# what do these output in the interpreter?
>>> val1 = triple_it_v1(2)
>>> val1 + 1
7
>>> val2 = triple_it_v2(2)
>>> val2 + 1
TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'
```

Python Tutor

```
def max(x, y):
    if x > y:
        return x
    else:
        return y
x = 3
y = 4 + max(17, x + 6) * 0.1
z = x / y
```

Doctests

- Write the docstring to explain what it does
    - What does the function return? What are corner cases for parameters?

```
def max(x, y):
    """Returns the larger value of arguments x and y
    >>> max(6, 0)
    6
    """

    return x if x > y else y
```

- Write doctest to show what it should do
    - Before you write the implementation.
    - python3 –m doctest [-v] file.py

Computational Structures in Data Science

Python: Control Flow

Berkeley
UNIVERSITY OF CALIFORNIA

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")
```

Live Coding Demo

```python
course = 'C88C'
time = '3:00'
if time == '2:00':
    print(f"Go to {course}")
else:
    print("Go get some ☕")


Go to C88C
```

Conditional Expression Shorthand

- Return a Value Based on some condition

```
<true expression> if <predicate> else <false expression>
```

- Example:

```
status = "it's hot!" if temperature > 85 else 'not hot…'
```

Computational Structures in Data Science

Iteration with `while` Loops

Berkeley
UNIVERSITY OF CALIFORNIA

Learning Objectives

- Use a while loop to repeat some task.
- Write an expression to control when a while loop stops executing

while Statement – Iteration Control

- Repeat a block of statements until a predicate expression is satisfied

```
<initialization statements>
while <predicate expression>:
    <body statements>

<rest of the program>
```

Sum The Numbers

- This is a task we'll see many times!

```
total = 0
n = 1
while n <= 10:
    total += n
    n += 1
print(total)
```

Computational Structures in Data Science

Iteration With `for` Loops

Berkeley
UNIVERSITY OF CALIFORNIA

Learning Objectives

- Compare a for loop and a while loop.
- Learn to use range()
- Use a string as a sequence of letters

- 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>
```

<sequence expression> — What's that?

- Sequences are a type of data that can broken down into smaller parts.
- Common sequences:
  - range() – give me all the numbers
  - Strings, e.g, "Hello, C88C!"
    - What is it a sequence of? Characters!
  - lists (next!)
- We'll start with two basic facts:
  - `range(10)` is the numbers 0 to 9, or range(0, 10)
  - [] means "indexing" an item in a sequence.
  - `"Hello"[0] == "H"`

Data-Driven Iteration

- describe an expression to perform on each item in a sequence
- let the data dictate the control

```
[ <expr with loop var> for <loop var> in <sequence expr > ]
```

Outro

- Today, we covered:

  - Python functions, if statements, conditional expressions, for/while loops, "data-driven iteration"

Outro

- At this point, you have seen enough of the language to read/write most of Python (in theory)

- Labs, homework, and self-study will give you the hands-on practice for Python syntax to become second-nature (aka a "craft")

- Keep up the great work, and have a nice weekend!