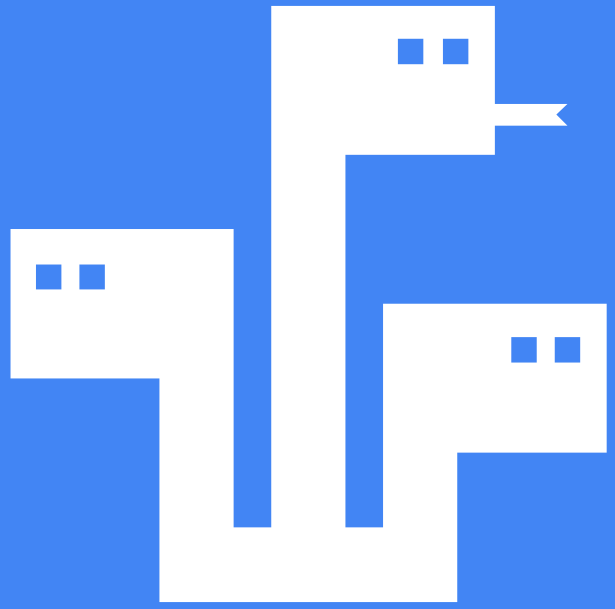
Control Flow

Python - Nick Reynolds
September 23, 2017



This Class

- Booleans
- Conditionals
- Functions

Booleans

True or False

Conditions

1. `5 < 10`
2. `10 != 5`
3. `"Bob" == "bob"`
4. `"AAA" == "AAA"`
5. `5 == 10 or 1 < 5`
6. `(3 > 2 and 1 == 1) or False`
7. `not True`
8. `True == False or not False`

Conditions

```
1.  5 < 10                              True
2.  10 != 5                             True
3.  "Bob" == "bob"                      False
4.  "AAA" == "AAA"                      True
5.  5 == 10 or 1 < 5                    True
6.  (3 > 2 and 1 == 1) or False         True
7.  not True                            False
8.  True == False or not False          True
```

Conditionals

Conditionals

- Conditionals or IF statements control the flow of your application
- If conditions are satisfied then code is executed, else it is not
- Keywords:
  - and
  - or
  - not

```
>>> a = 5
>>> if a < 10:
...     print('a is small!')
... else:
...     print('a is big!')
...
a is small!
>>> if a < 10 and a == 3:
...     print('a is 3!')
... else:
...     print('no luck!')
...
no luck!
```

Conditionals Continued

- Simply checking for a condition to be true, if true do x
- Key words
  - If
  - Elif (else if)
  - else

```
>>> x = 5
>>> if x != 10:
...     print("Hello!")
Hello
>>> if x < 5:
...     print("Hello!")
... elif x == 5:
...     print("World!")
... else:
...     print("Me")
World!
```

Conditionals Continued

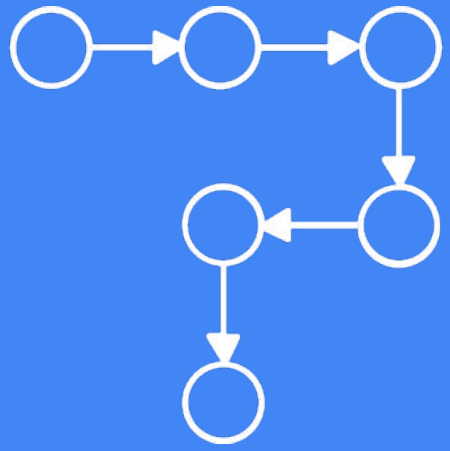
```
>>> a = 5
>>> b = 10
>>> if a < 10 and b > a:
...     print('fizz')
... elif a == 3 or b <= 3:
...     print('buzz')
... else:
...     print('pop')
...
fizz
```

```
>>> shop = ["apple", "banana"]
>>> b = 10
>>> if "apple" in shop and b != 10:
...     print('fizz')
... elif "pear" not in shop:
...     print('buzz')
... else:
...     print('pop')
...
buzz
```

Pen and Paper

Checkpoint

Functions



Functions

- A procedure is a named block of statements that performs an operation. Instead of writing it each time
- *Why would we want this?*

```
def hodor():  # Define a procedure named hodor
    print("hodor, hodor")
    print("hodor")

hodor()  # Call our hodor procedure

hodor()  # Call it again
```

Functions

- Parameters are the inputs to your procedure
- Lets us make more general functions
- *Why would we want this?*

```
def say_it(word):  # Define a procedure named say_it
    print("{} {}".format(word, word))
    print(word)

say_it("Hey")  # Call our hodor procedure

say_it()  # Call it again
```

Returning Values

- Functions can return values using the return keyword, this means that the function evaluates to this value

```
>>> def add(x, y):
...     result = x + y
...     return result

>>> add(1,3)
4
>>> add(2, add(5, 6))
13
>>> x = add(1,3)
>>> x + 4
8
```

Testing (Made easy!)

- Python has a built in tester, doctests
- Write function calls and the expected result on the next line, run the tests!

```
>>> def add(x, y):
...     """
...     >>> add(3, 4)
...     7
...     """
...     result = x + y
...     return result

>>> import doctest
>>> doctest.testmod()
```

Practical

References

- http://pwp.stevecassidy.net/python/more-python.html
- https://thenounproject.com/