hello, section!

week 6

pset 5 recap

why Python?

different languages

=

different tradeoffs

Python vs. C

Python vs. C

high level vs. low level

Python vs. C

interpreted vs. compiled

dynamically vs. statically typed

Python vs. C

object oriented

Python vs. C

easy to write
flexibility
wider communities

Python

- → high level
- → easier to write
- → interpreted
- → dynamically typed
- → automatic memory management

C

- → low level
- → more difficult to write
- → compiled
- → statically typed
- → manual memory management

syntax

syntax

whitespace

{
}

syntax

```
:                    {
:                    }
```

syntax

```
while (True):
    ...
```

```
while (true)
{
    ...
}
```

variables

variables

number

string

list

tuple

dictionary

variables

no type declaration necessary

Python

```
i = 5
s = "hello!"
```

C

```
int i = 5;
string s = "hello!";
```

Python

```
import cs50

s = cs50.get_string()
if s != None:
    for c in s:
        print(c)
```

C

```
#include <cs50.h>
#include <stdio.h>
#include <string.h>

int main(void)
{
    string s = get_string();
    if (s != NULL)
    {
        for (int i = 0; i < strlen(s); i++)
        {
            printf("%c\n", s[i]);
        }
    }
}
```

Python

```
import cs50

f = cs50.get_float()
c = 5.0 / 9.0 * (f - 32.0)
print(f"{c:.1f}")
```

C

```
#include <cs50.h>
#include <stdio.h>

int main(void)
{
    float f = get_float();
    float c = 5.0 / 9.0 * (f - 32.0);
    printf("%.1f\n", c);
}
```

conditionals

Python	C
`or`	`||`
`and`	`&&`

Python

```
x = cs50.get_int()
if x < 0:
    print("x is negative")
elif x > 0:
    print("x is positive")
else:
    print("x is zero")
```

C

```
int x = get_int();
if (x < 0)
{
    printf("x is negative\n");
}
else if (x > 0)
{
    printf("x is positive\n");
}
else
{
    printf("x is zero\n");
}
```

loops

Python

```python
i = 0
while i < 100:
    print(i)
    i += 1

for j in range(0, 101, 2):
    print(j)
```

C

```c
int i = 0;
while(i < 100)
{
    printf("%i\n", ++i);
}

for(int j = 0; j < 100; j += 2)
{
    printf("%i\n", j);
}
```

command line input

Python

```
import sys

for i in range(len(sys.argv)):
    print(sys.argv[i])
```

C

```
#include <cs50.h>
#include <stdio.h>

int main(int argc, string argv[])
{
    for (int i = 0; i < argc; i++)
    {
        printf("%s\n", argv[i]);
    }
}
```

functions

functions

- no main function necessary
- no function prototype necessary
- can return multiple values

Python

```
def square(x):
    return x ** 2

base = cs50.get_float()
print(square(base))
```

C

```
float square(float x);

int main(void)
{
    float base = get_float();
    printf("%f\n", square(base));
    return 0;
}

float square(float x)
{
    return x * x;
}
```

Python

```python
def main():
    for i in range(3):
        cough()

def cough():
    print("cough")

if __name__ == "__main__":
    main()
```

C

```c
#include <stdio.h>

void cough(void);

int main(void)
{
    for (int i = 0; i < 3; i++)
    {
        cough();
    }
}

void cough(void)
{
    printf("cough\n");
}
```

objects

objects

- similar to C structs
- define methods & properties
- objects are instances of a class

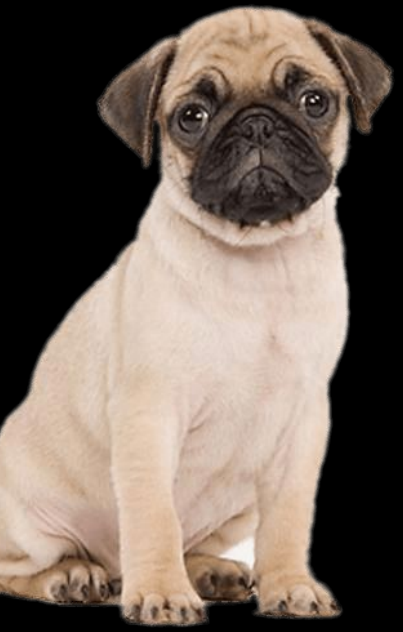
objects

```
class Dog():

  def __init__(self, breed, good=True):
    self.breed = breed
    self.good = good

  def bark():
    print("ruff")
```

objects



```
fido = Dog("golden")
fido.bark()

pepper = Dog("pug", False)
pepper.bark()
```

Python is *object-oriented*

almost everything (underneath the hood) is an object!

new Python data types

tuples
lists
dictionaries

tuples

```
coordinate = (2, 3)
```

- ordered, immutable data
- iterable
- any length

tuples

```
coordinate = (2, 3)
print(coordinate[0])
print(coordinate[1])
```

tuples

```
coordinate = (2, 3)
x, y = coordinate
print(x)
print(y)
```

tuples

```
coordinates = [(2, 3), (-1, 4), (0, 6)]

for x,y in coordinates:
    print("x: %i, y: %i" % (x, y))
```

arrays in C → lists in Python

- ➔ fixed length
- ➔ values of one type
- ➔ direct blocks of memory

- ➔ flexible length
- ➔ values of mixed data types
- ➔ objects

Python

```
grades = [87, 61, 90, 83,
78, 99, 93, 55, 81, 76]

# SORT
grades.sort()

# LEGAL
grades.append(100)

# PRINT GRADES
for grade in grades:
    print(grade)
```

C

```
int grades[10] = {87, 61, 90,
83, 78, 99, 93, 55, 81, 76};

// ILLEGAL
grades[10] = 100;

// PRINT GRADES
for (int k = 0; k < 11; k++)
{
    printf("%i\n", grades[k]);
}
```

Python

```
grades = [87, 61, 90, 83,
78, 99, 93, 55, 81, 76]

# SORT
grades.sort()

# LEGAL
grades.append(100)

# PRINT GRADES
for grade in grades:
    print(grade)
```

C

```
int grades[10] = {87, 61, 90,
83, 78, 99, 93, 55, 81, 76};

// ILLEGAL
grades[10] = 100;

// PRINT GRADES
for (int k = 0; k < 11; k++)
{
    printf("%i\n", grades[k]);
}
```

lists

```
x = []
x.append(1)
x.extend([2, 4])
x.insert(2, 3)
x.remove(4)
x.sort()
len(x)
```

...

dictionary

- key, value pairs
- mutable
- use for associated data
- like a hash table in C

```
pizzas = {
        "cheese": 9,
        "pepperoni": 10,
        "vegetable": 11,
        "buffalo chicken": 12
    }
```

dictionary

brackets denote dicts

```
pizzas = {
    "cheese": 9,
    "pepperoni": 10,
    "vegetable": 11,
    "buffalo chicken": 12
}
```

dictionary

keys

```
pizzas = {
    "cheese": 9,
    "pepperoni": 10,
    "vegetable": 11,
    "buffalo chicken": 12
}
```

dictionary

colons attach keys to values

```
pizzas = {
    "cheese": 9,
    "pepperoni": 10,
    "vegetable": 11,
    "buffalo chicken": 12
}
```

dictionary

values

```
pizzas = {
    "cheese": 9,
    "pepperoni": 10,
    "vegetable": 11,
    "buffalo chicken": 12
}
```

dictionary

commas separate key value pairs

```
pizzas = {
    "cheese": 9,
    "pepperoni": 10,
    "vegetable": 11,
    "buffalo chicken": 12
}
```

dictionary

```
pizzas = {
      "cheese": 9,
      "pepperoni": 10,
      "vegetable": 11,
      "buffalo chicken": 12
   }
pizzas["cheese"] = 8
print(pizzas["cheese"])
```

dictionary

```
pizzas = {
      "cheese": 9,
      "pepperoni": 10,
      "vegetable": 11,
      "buffalo chicken": 12
   }

for key, value in pizzas:
    print(key + " costs " + "$" + str(value))
```

dictionary

```
x = {}
x.update({"cheese": 8})
x.keys()
x.values()
x.clear()
```

...

pset 6 tips & tricks

- Google syntax questions
- look online for available functions in Python
- we are using **Python 3**
- use the terminal to test things out