Este é o CS50

Introdução à Ciência da Computação (CS50)

OpenCourseWare

Doar (https://cs50.harvard.edu/donate)

David J. Malan (https://cs.harvard.edu/malan/)
malan@harvard.edu
(https://www.facebook.com/dmalan) (https://github.com/dmalan)
(https://www.instagram.com/davidjmalan/) (https://www.linkedin.com/in/malan/)
(https://www.reddit.com/user/davidjmalan)
(https://www.threads.net/@davidjmalan) (https://twitter.com/davidjmalan)

Aula 6

Bem-vindo!

- Nas semanas anteriores, você foi apresentado aos elementos fundamentais da programação.
- Você aprendeu sobre programação em uma linguagem de programação de baixo nível chamada C.
- Hoje, vamos trabalhar com uma linguagem de programação de alto nível chamada *Python* .
- Ao aprender essa nova linguagem, você perceberá que será mais capaz de aprender novas linguagens de programação por conta própria.

Olá, Python!

- Ao longo das décadas, os seres humanos perceberam como as decisões de design tomadas em linguagens de programação anteriores poderiam ser aprimoradas.
- Python é uma linguagem de programação que se baseia no que você já aprendeu em C.
- Além disso, o Python tem acesso a um grande número de bibliotecas criadas pelos usuários.
- Diferentemente de C, que é uma *linguagem compilada* , Python é uma *linguagem interpretada* , onde você não precisa compilar seu programa separadamente. Em vez disso, você executa seu programa no *interpretador Python* .
- Até este ponto, o código tinha a seguinte aparência:

```c
// A program that says hello to the world

#include <stdio.h>

int main(void)
{
    printf("hello, world\n");
}
```

- Hoje em dia, você perceberá que o processo de escrever e compilar código foi simplificado.
- Por exemplo, o código acima será renderizado em Python da seguinte forma:

```python
# A program that says hello to the world

print("hello, world")
```

Observe que o ponto e vírgula desapareceu e que nenhuma biblioteca é necessária. Você pode executar este programa no seu terminal digitando `python hello.py`.

- Notavelmente, o Python consegue implementar com relativa simplicidade o que era bastante complicado em C.

soletrador

- Para ilustrar essa simplicidade, vamos digitar 'code dictionary.py' na janela do terminal e escrever o código da seguinte forma:

```
# Words in dictionary
words = set()


def check(word):
    """Return true if word is in dictionary else false"""
    return word.lower() in words


def load(dictionary):
    """Load dictionary into memory, returning true if successful else false"""
    with open(dictionary) as file:
        words.update(file.read().splitlines())
    return True


def size():
    """Returns number of words in dictionary if loaded else 0 if not yet loade
    return len(words)


def unload():
    """Unloads dictionary from memory, returning true if successful else false
    return True
```

Observe que existem quatro funções acima. Na `check` função `add_word`, se `a` `word` estiver em `a` `words`, ela retorna `a` `True`. É muito mais simples do que uma implementação em C! Da mesma forma, na `load` função `add_word`, o arquivo de dicionário é aberto. Para cada linha nesse arquivo, adicionamos essa linha a `a` `words`. Usando `rstrip` `add_word`, a quebra de linha final é removida da palavra adicionada. `add_word` `size` simplesmente retorna o `len` comprimento de `a` `words`. `add_word` `unload` só precisa retornar `a` `True` porque o Python gerencia a memória por conta própria.

- O código acima ilustra por que existem linguagens de alto nível: para simplificar e permitir que você escreva código com mais facilidade.
- No entanto, a velocidade tem um preço. Como o C permite que você, o programador, tome decisões sobre o gerenciamento de memória, ele pode ser executado mais rapidamente do que o Python — dependendo do seu código. Enquanto o C executa apenas as suas

linhas de código, o Python executa todo o código subjacente quando você chama as funções internas do Python.

- Você pode aprender mais sobre funções na documentação do Python. (https://docs.python.org/3/library/functions.html)

Filtro

- Para ilustrar ainda mais essa simplicidade, crie um novo arquivo digitando-o `code blur.py` na janela do terminal e escreva o código da seguinte forma:

```
# Blurs an image

from PIL import Image, ImageFilter

# Blur image
before = Image.open("bridge.bmp")
after = before.filter(ImageFilter.BoxBlur(1))
after.save("out.bmp")
```

  Observe que este programa importa módulos `Image` de `ImageFilter` uma biblioteca chamada `PIL`. Ele recebe um arquivo de entrada e cria um arquivo de saída.

- Além disso, você pode criar um novo arquivo com `edges.py` o seguinte nome:

```
# Finds edges in an image

from PIL import Image, ImageFilter

# Find edges
before = Image.open("bridge.bmp")
after = before.filter(ImageFilter.FIND_EDGES)
after.save("out.bmp")
```

  Note que este código é um pequeno ajuste ao seu `blur` código, mas produz um resultado drasticamente diferente.

- Python permite abstrair a programação que seria muito mais complicada em C e outras linguagens de programação *de baixo nível* .

Funções

- Em C, você pode ter visto funções como as seguintes:

```
printf("hello, world\n");
```

- Em Python, você verá funções como as seguintes:

```
print("hello, world")
```

Bibliotecas, módulos e pacotes

- Assim como em C, a biblioteca CS50 pode ser utilizada em Python.
- As seguintes funções serão particularmente úteis:

```
get_float
get_int
get_string
```

- Você pode importar a biblioteca cs50 da seguinte forma:

```
import cs50
```

- Você também tem a opção de importar apenas funções específicas da biblioteca CS50, da seguinte forma:

```
from cs50 import get_float, get_int, get_string
```

Cordas

- Em C, você talvez se lembre deste código:

```
// get_string and printf with %s

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

int main(void)
{
    string answer = get_string("What's your name? ");
    printf("hello, %s\n", answer);
}
```

- Este código é transformado em Python para:

```
# get_string and print, with concatenation

from cs50 import get_string

answer = get_string("What's your name? ")
print("hello, " + answer)
```

  Você pode escrever este código executando-o `code hello.py` na janela do terminal. Em seguida, você pode executar este código executando o comando `python hello.py`. Observe como o `+` sinal concatena `"hello, "` os dois pontos `answer`.

- Da mesma forma, isso pode ser feito sem concatenação:

```
# get_string and print, without concatenation

from cs50 import get_string
```

```
answer = get_string("What's your name? ")
print("hello,", answer)
```

Observe que a instrução print cria automaticamente um espaço entre a `hello` instrução e o ponto `answer`.

- Da mesma forma, você poderia implementar o código acima como:

```
# get_string and print, with format strings

from cs50 import get_string

answer  = get_string("What's your name? ")
print(f"hello, {answer}")
```

Observe como as chaves permitem que a `print` função interpole o `answer` texto que `answer` aparece dentro delas. O uso de chaves `f` é necessário para incluir a `answer` formatação adequada.

Parâmetros posicionais e parâmetros nomeados

- Funções em C como `f` `fread`, `fwrite` `g` e ` `printf` f` usam argumentos posicionais, onde você fornece argumentos separados por vírgulas. Você, o programador, deve lembrar qual argumento está em qual posição. Esses são chamados de *argumentos posicionais* .
- Em Python, *parâmetros nomeados* permitem fornecer argumentos independentemente da posição dos mesmos.
- Você pode aprender mais sobre os parâmetros da `print` função na documentação (https://docs.python.org/3/library/functions.html#print) .
- Ao acessar essa documentação, você poderá ver o seguinte:

```
print(*objects, sep=' ', end='\n', file=None, flush=False)
```

Observe que vários objetos podem ser fornecidos para impressão. Um separador de espaço simples é fornecido e será exibido quando mais de um objeto for fornecido `print`. Da mesma forma, uma nova linha é fornecida ao final da `print` instrução.

Variáveis

- A declaração de variáveis também foi simplificada. Em C, você poderia ter ` `int counter = 0;` . Em Python, essa mesma linha seria ` `counter = 0` . Você não precisa declarar o tipo da variável.
- `counter += 1` Em Python , o incremento é preferencial em um, perdendo a capacidade encontrada em C de digitar `counter++` .

Tipos

- Em Python, os tipos de dados não precisam ser declarados explicitamente. Por exemplo, você viu que o valor `answer` acima é uma string, mas não precisamos informar isso ao interpretador: ele já sabia disso por conta própria.
- Em Python, os tipos comumente usados incluem:

  ```
  bool
  float
  int
  str
  ```

  Note que `x` `long` e `y` `double` estão faltando. O Python cuidará de qual tipo de dado deve ser usado para números maiores e menores.
- Outros tipos de dados em Python incluem:

  ```
  range   sequence of numbers
  list    sequence of mutable values
  tuple   sequence of immutable values
  dict    collection of key-value pairs
  set     collection of unique values
  ```
- Cada um desses tipos de dados pode ser implementado em C, mas em Python, eles podem ser implementados de forma mais simples.

Calculadora

- Você deve se lembrar `calculator.c` de algo que vimos anteriormente no curso:

  ```c
  // Addition with int

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

  int main(void)
  {
      // Prompt user for x
      int x = get_int("x: ");

      // Prompt user for y
      int y = get_int("y: ");

      // Perform addition
      printf("%i\n", x + y);
  }
  ```
- Podemos implementar uma calculadora simples, assim como fizemos em C. `code calculator.py` Digite o seguinte código na janela do terminal:

```
# Addition with int [using get_int]

from cs50 import get_int

# Prompt user for x
x = get_int("x: ")

# Prompt user for y
y = get_int("y: ")

# Perform addition
print(x + y)
```

Observe como a biblioteca CS50 é importada. Em seguida, os valores de ` `x` x` e `y` `y` são obtidos do usuário. Finalmente, o resultado é impresso. Note que a `main` função que seria vista em um programa em C desapareceu completamente! Embora seja possível utilizar uma `main` função, ela não é necessária.

- É possível remover as limitações da biblioteca CS50. Modifique seu código da seguinte forma:

```
# Addition with int [using input]

# Prompt user for x
x = input("x: ")

# Prompt user for y
y = input("y: ")

# Perform addition
print(x + y)
```

Observe como a execução do código acima resulta em um comportamento estranho do programa. Por que isso acontece?

- Você deve ter adivinhado que o interpretador entendeu ` a` `x` e `y` `b` como strings. Você pode corrigir seu código empregando a `int` função da seguinte forma:

```
# Addition with int [using input]

# Prompt user for x
x = int(input("x: "))

# Prompt user for y
y = int(input("y: "))

# Perform addition
print(x + y)
```

Observe como a entrada para ` `x` a` e `b` `y` é passada para a `int` função, que a converte em um número inteiro. Sem converter ` `x` a` e `y` `b` para inteiros, os caracteres seriam concatenados.

Condicionais

- Em C, você talvez se lembre de um programa como este:

```c
// Conditionals, Boolean expressions, relational operators

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

int main(void)
{
    // Prompt user for integers
    int x = get_int("What's x? ");
    int y = get_int("What's y? ");

    // Compare integers
    if (x < y)
    {
        printf("x is less than y\n");
    }
    else if (x > y)
    {
        printf("x is greater than y\n");
    }
    else
    {
        printf("x is equal to y\n");
    }
}
```

- Em Python, ficaria assim:

```python
# Conditionals, Boolean expressions, relational operators

from cs50 import get_int

# Prompt user for integers
x = get_int("What's x? ")
y = get_int("What's y? ")

# Compare integers
if x < y:
    print("x is less than y")
elif x > y:
    print("x is greater than y")
else:
    print("x is equal to y")
```

  Observe que não há mais chaves. Em vez disso, são utilizadas indentações. Em segundo lugar, utiliza-se um dois-pontos na `if` declaração. Além disso, `elif` substitui `else if`. Os parênteses também não são mais necessários nas declarações `if` e . `elif`

- Analisando mais detalhadamente as comparações, considere o seguinte código em C:

```c
// Logical operators
```

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

int main(void)
{
    // Prompt user to agree
    char c = get_char("Do you agree? ");

    // Check whether agreed
    if (c == 'Y' || c == 'y')
    {
        printf("Agreed.\n");
    }
    else if (c == 'N' || c == 'n')
    {
        printf("Not agreed.\n");
    }
}
```

- O exposto acima pode ser implementado da seguinte forma:

```python
# Logical operators

from cs50 import get_string

# Prompt user to agree
s = get_string("Do you agree? ")

# Check whether agreed
if s == "Y" or s == "y":
    print("Agreed.")
elif s == "N" or s == "n":
    print("Not agreed.")
```

Observe que as duas barras verticais utilizadas em C foram substituídas por `or`. De fato, muitas pessoas preferem Python por ser mais legível para humanos. Observe também que `char` não existe em Python. Em vez disso, `str` são utilizadas .

- Outra abordagem para esse mesmo código poderia ser a seguinte, utilizando *listas* :

```python
# Logical operators, using lists

from cs50 import get_string

# Prompt user to agree
s = get_string("Do you agree? ")

# Check whether agreed
if s in ["y", "yes"]:
    print("Agreed.")
elif s in ["n", "no"]:
    print("Not agreed.")
```

Observe como podemos expressar várias palavras-chave como `y` e `yes` em um `list`.

Programação Orientada a Objetos

- É possível que certos tipos de valores não apenas possuam propriedades ou atributos, mas também funções. Em Python, esses valores são conhecidos como *objetos*.
- Em C, podíamos criar um tipo de dados `struct` onde era possível associar múltiplas variáveis dentro de um único tipo de dados criado pelo usuário. Em Python, podemos fazer isso e também incluir funções em um tipo de dados criado pelo usuário. Quando uma função pertence a um *objeto* específico , ela é conhecida como um *método* .
- Por exemplo, em Python, existem *métodos* `strs` integrados . Portanto, você poderia modificar seu código da seguinte forma:

```
# Logical operators, using lists

# Prompt user to agree
s = input("Do you agree? ").lower()

# Check whether agreed
if s in ["y", "yes"]:
    print("Agreed.")
elif s in ["n", "no"]:
    print("Not agreed.")
```

  Observe como o valor antigo de `s` é sobrescrito com o resultado de `s.lower()`, um método embutido de `strs`.

- Da mesma forma, você deve se lembrar de como copiávamos uma string em C:

```
// Capitalizes a copy of a string without memory errors

#include <cs50.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{
    // Get a string
    char *s = get_string("s: ");
    if (s == NULL)
    {
        return 1;
    }

    // Allocate memory for another string
    char *t = malloc(strlen(s) + 1);
    if (t == NULL)
    {
        return 1;
    }

    // Copy string into memory
    strcpy(t, s);

    // Capitalize copy
    if (strlen(t) > 0)
    {
```

```
        t[0] = toupper(t[0]);
    }

    // Print strings
    printf("s: %s\n", s);
    printf("t: %s\n", t);

    // Free memory
    free(t);
    return 0;
}
```

Observe o número de linhas de código.

- Podemos implementar o acima em Python da seguinte forma:

```
# Capitalizes a copy of a string

# Get a string
s = input("s: ")

# Capitalize copy of string
t = s.capitalize()

# Print strings
print(f"s: {s}")
print(f"t: {t}")
```

Observe como este programa é muito mais curto do que sua contraparte em C.

- Nesta aula, vamos apenas abordar superficialmente o Python. Portanto, a documentação do Python (https://docs.python.org) será de particular importância à medida que você prosseguir.
- Você pode aprender mais sobre métodos de string na documentação do Python. (https://docs.python.org/3/library/stdtypes.html#string-methods)

Laços

- Os laços de repetição em Python são muito semelhantes aos de C. Você pode se lembrar do seguinte código em C:

```
// Demonstrates for loop

#include <stdio.h>

int main(void)
{
    for (int i = 0; i < 3; i++)
    {
        printf("meow\n");
    }
}
```

- `for` Os loops podem ser implementados em Python da seguinte forma:

```
# Better design

for i in range(3):
    print("meow")
```

Note que `i` nunca é usado explicitamente. No entanto, o Python incrementará o valor de `i`.

- Além disso, um `while` loop poderia ser implementado da seguinte forma:

```
# Demonstrates while loop

i = 0
while i < 3:
    print("meow")
    i += 1
```

- Para aprofundar nosso entendimento sobre loops e iterações em Python, vamos criar um novo arquivo com `uppercase.py` o seguinte nome:

```
# Uppercases string one character at a time

before = input("Before: ")
print("After:  ", end="")
for c in before:
    print(c.upper(), end="")
print()
```

Observe como ` `end=` is` é usado para passar um parâmetro para a `print` função que continua a linha sem uma quebra de linha. Este código passa uma string por vez.

- Lendo a documentação, descobrimos que o Python possui métodos que podem ser implementados em toda a string, da seguinte forma:

```
# Uppercases string all at once

before = input("Before: ")
after = before.upper()
print(f"After:  {after}")
```

Observe como isso `.upper` é aplicado à string inteira.

Abstração

- Como mencionamos anteriormente, você pode aprimorar ainda mais o nosso código usando funções e abstraindo várias partes do código em funções. Modifique o `meow.py` código que você criou anteriormente da seguinte forma:

```
# Abstraction

def main():
    for i in range(3):
        meow()
```

```
# Meow once
def meow():
    print("meow")


main()
```

Observe que a `meow` função abstrai a `print` instrução. Além disso, note que a `main` função aparece no início do arquivo. No final do arquivo, a `main` função é chamada. Por convenção, espera-se que você crie uma `main` função em Python.

- De fato, podemos passar variáveis entre nossas funções da seguinte maneira:

```
# Abstraction with parameterization

def main():
    meow(3)


# Meow some number of times
def meow(n):
    for i in range(n):
        print("meow")


main()
```

Observe como `meow` agora a função recebe uma variável `n`. Dentro da `main` função, você pode chamá-la `meow` e passar um valor como esse `3` para ela. Em seguida, `meow` a função utiliza o valor de `x` `n` no `for` loop.

- Ao ler o código acima, observe como você, como programador C, consegue compreendê-lo com bastante facilidade. Embora algumas convenções sejam diferentes, os fundamentos que você aprendeu anteriormente são muito evidentes nesta nova linguagem de programação.

Truncamento e Imprecisão de Ponto Flutuante

- Lembre-se de que, em C, experimentamos o truncamento, onde a divisão de um número inteiro por outro pode resultar em um resultado impreciso.
- Você pode ver como o Python lida com essa divisão modificando seu código da seguinte forma `calculator.py`:

```
# Division with integers, demonstration lack of truncation

# Prompt user for x
x = int(input("x: "))

# Prompt user for y
y = int(input("y: "))

# Divide x by y
```

```
z = x / y
print(z)
```

Note que a execução deste código resulta em um valor, mas que, se você observasse mais dígitos posteriormente, `.333333` perceberia que estamos diante de uma *imprecisão de ponto flutuante* . O truncamento não ocorre.

- Podemos revelar essa imprecisão modificando ligeiramente nossos códigos:

```
# Floating-point imprecision

# Prompt user for x
x = int(input("x: "))

# Prompt user for y
y = int(input("y: "))

# Divide x by y
z = x / y
print(f"{z:.50f}")
```

Note que este código revela a imprecisão. O Python ainda enfrenta esse problema, assim como o C.

Exceções

- Vamos explorar mais sobre as exceções que podem ocorrer ao executarmos código Python.
- Modifique `calculator.py` da seguinte forma:

```
# Doesn't handle exception

# Prompt user for an integer
n = int(input("Input: "))
print("Integer")
```

Observe que inserir dados incorretos pode resultar em um erro.

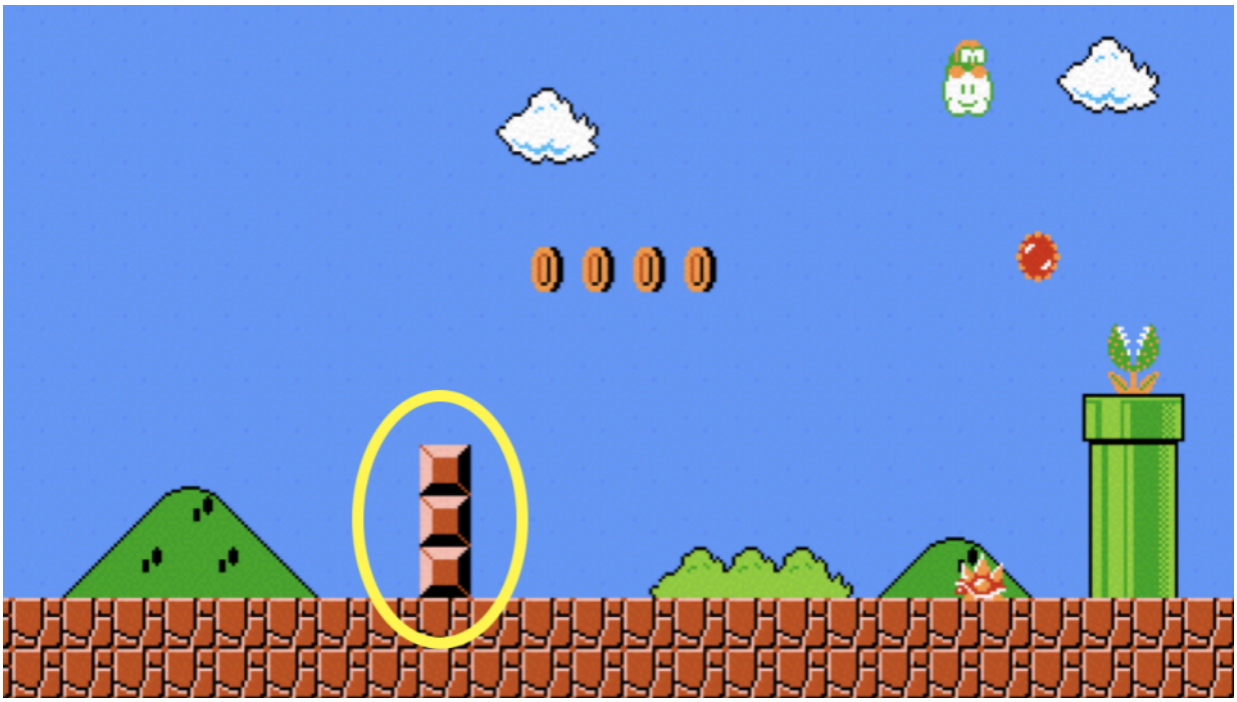
- Podemos `try` lidar com possíveis exceções e *capturá-las* modificando nosso código da seguinte forma:

```
# Handles exception

# Prompt user for an integer
try:
    n = int(input("Input: "))
    print("Integer.")
except ValueError:
    print("Not integer.")
```

Observe que o código acima tenta repetidamente obter o tipo correto de dados, fornecendo solicitações adicionais quando necessário.

Mario

- Relembram o nosso desafio de algumas semanas atrás, de construir três blocos um em cima do outro, como no Mario?



- Em Python, podemos implementar algo semelhante a isto da seguinte forma:

```
# Prints a column of 3 bricks with a loop

for i in range(3):
    print("#")
```

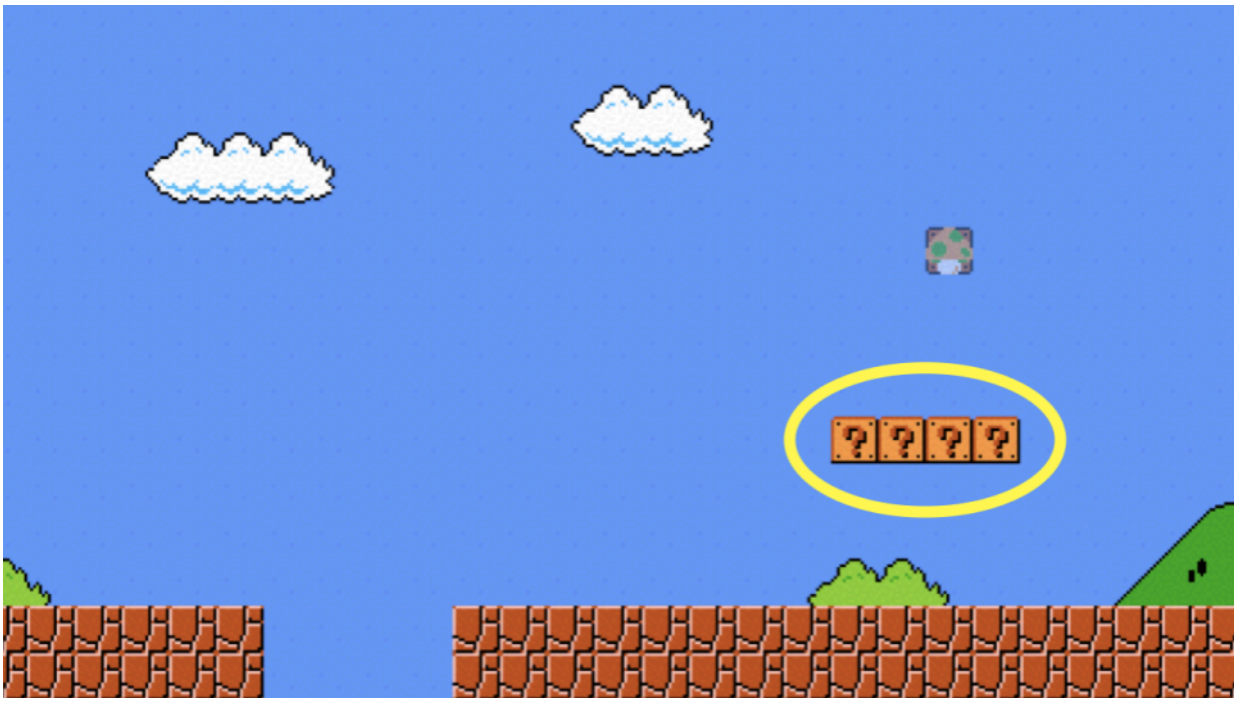
Isso imprime uma coluna de três tijolos.

- Em C, tínhamos a vantagem de um `do-while` laço de repetição. No entanto, em Python, é convencional utilizar um `while` laço de repetição, já que Python não possui um `do-while` laço nativo. Você pode escrever o código da seguinte forma em um arquivo chamado `mario.py`:

```
# Prints a column of n bricks with a loop

from cs50 import get_int

while True:
    n = get_int("Height: ")
    if n > 0:
        break

for i in range(n):
    print("#")
```

Observe como o laço while é usado para obter a altura. Assim que uma altura maior que zero é inserida, o laço é interrompido.

- Considere a seguinte imagem:



- Em Python, poderíamos implementar isso modificando seu código da seguinte forma:

```
# Prints a row of 4 question marks with a loop

for i in range(4):
    print("?", end="")
print()
```

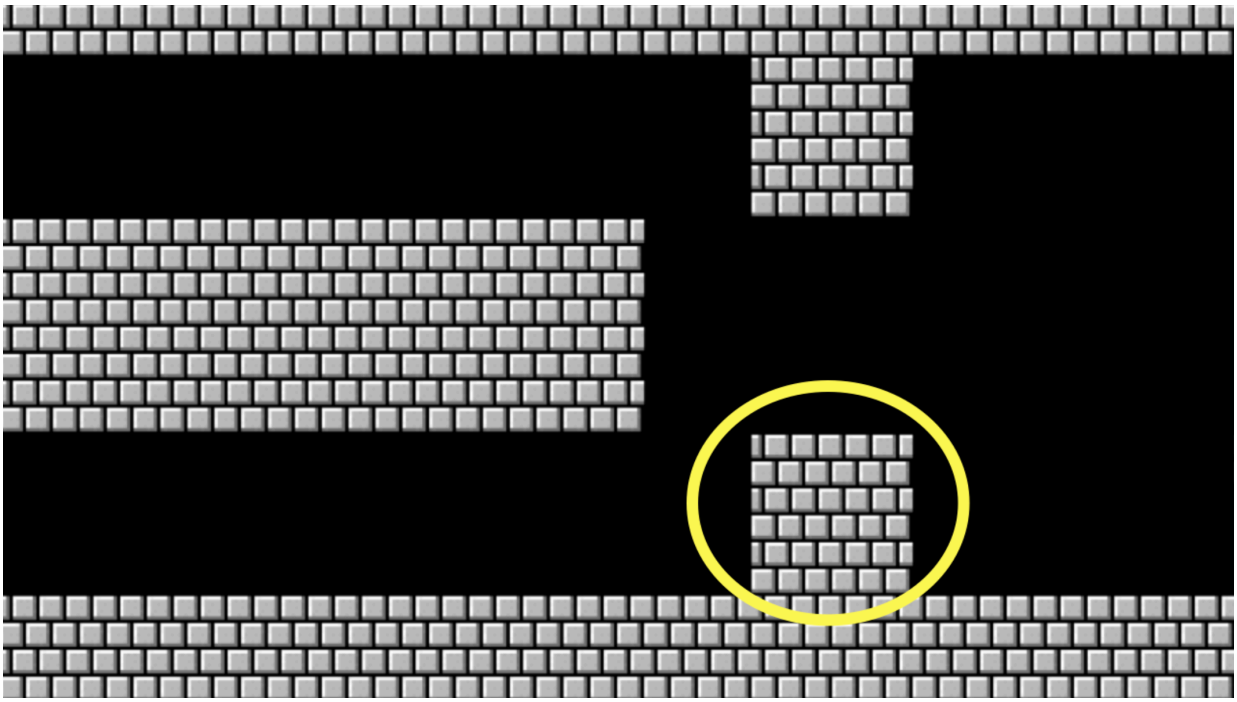
  Observe que você pode alterar o comportamento da `print` função para que ela permaneça na mesma linha da impressão anterior.

- De forma semelhante às versões anteriores, podemos simplificar ainda mais este programa:

```
# Prints a row of 4 question marks without a loop

print("?" * 4)
```

  Observe que podemos usar isso `*` para multiplicar a instrução print e repeti-la várias `4` vezes.

- Que tal um grande bloco de tijolos?



- Para implementar o que foi descrito acima, você pode modificar seu código da seguinte forma:

```
# Prints a 3-by-3 grid of bricks with loops

for i in range(3):
    for j in range(3):
        print("#", end="")
    print()
```

  Observe como um `for` laço existe dentro de outro. A `print` instrução adiciona uma nova linha ao final de cada linha de blocos.

- Você pode aprender mais sobre a `print` função na documentação do Python. (https://docs.python.org/3/library/functions.html#print)

Listas

- `list`s são uma estrutura de dados em Python.
- `list` Os objetos possuem métodos ou funções integrados.
- Por exemplo, considere o seguinte código:

```
# Averages three numbers using a list

# Scores
scores = [72, 73, 33]

# Print average
average = sum(scores) / len(scores)
print(f"Average: {average}")
```

  Observe que você pode usar o método integrado `sum` para calcular a média.

- Você pode até mesmo utilizar a seguinte sintaxe para obter valores do usuário:

```
# Averages three numbers using a list and a loop

from cs50 import get_int

# Get scores
scores = []
for i in range(3):
    score = get_int("Score: ")
    scores.append(score)

# Print average
average = sum(scores) / len(scores)
print(f"Average: {average}")
```

Observe que este código utiliza o método integrado `append` para listas.

- Você pode aprender mais sobre listas na documentação do Python. (https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range)
- Você também pode aprender mais sobre isso `len` na documentação do Python. (https://docs.python.org/3/library/functions.html#len)

Pesquisa e dicionários

- Também podemos pesquisar dentro de uma estrutura de dados.
- Considere um programa `phonebook.py` com o seguinte nome:

```
# Implements linear search for names using loop

# A list of names
names = ["Yuliia", "David", "John"]

# Ask for name
name = input("Name: ")

# Search for name
for n in names:
    if name == n:
        print("Found")
        break
else:
    print("Not found")
```

Observe como isso implementa uma busca linear para cada nome.

- No entanto, não precisamos iterar por uma lista. Em Python, podemos executar uma busca linear da seguinte forma:

```
# Implements linear search for names using `in`

# A list of names
names = ["Yuliia", "David", "John"]

# Ask for name
name = input("Name: ")
```

```
# Search for name
if name in names:
    print("Found")
else:
    print("Not found")
```

Observe como `in` é utilizado para implementar a busca linear.

- Ainda assim, esse código poderia ser melhorado.
- Lembre-se que um *dicionário* é `dict` uma coleção de pares *chave* - *valor* .
- Você pode implementar um dicionário em Python da seguinte forma:

```
# Implements a phone book as a list of dictionaries, without a variable

from cs50 import get_string

people = [
    {"name": "Yuliia", "number": "+1-617-495-1000"},
    {"name": "David", "number": "+1-617-495-1000"},
    {"name": "John", "number": "+1-949-468-2750"},
]

# Search for name
name = get_string("Name: ")
for person in people:
    if person["name"] == name:
        print(f"Found {person['number']}")
        break
else:
    print("Not found")
```

Observe que o dicionário é implementado tendo ambos os parâmetros `name` `and` `number` para cada entrada.

- Melhor ainda, falando estritamente, não precisamos de ambos, `a` `name` e `a` `number` . Podemos simplificar este código da seguinte forma:

```
# Implements a phone book using a dictionary

from cs50 import get_string

people = {
    "Yuliia": "+1-617-495-1000",
    "David": "+1-617-495-1000",
    "John": "+1-949-468-2750",
}

# Search for name
name = get_string("Name: ")
if name in people:
    print(f"Number: {people[name]}")
else:
    print("Not found")
```

Observe que o dicionário é implementado usando chaves. Em seguida, a instrução `if name in people` busca para verificar se o valor de `people` `name` está presente no `people` dicionário. Além disso, observe como, na `print` instrução, podemos acessar o dicionário `people` usando o valor de `people` `name`. Muito útil!

- O Python fez o possível para atingir um *tempo de execução constante* usando suas funções de busca integradas.
- Você pode aprender mais sobre dicionários na documentação do Python. (https://docs.python.org/3/library/stdtypes.html#dict)

Argumentos da linha de comando

- Assim como em C, você também pode utilizar argumentos de linha de comando. Considere o seguinte código:

```
# Prints a command-line argument

from sys import argv

if len(argv) == 2:
    print(f"hello, {argv[1]}")
else:
    print("hello, world")
```

Observe que o `argv[1]` texto é impresso usando uma *string formatada* , indicada pelo `<p>f` ` presente na `print` declaração.

- Você pode aprender mais sobre a `sys` biblioteca na documentação do Python. (https://docs.python.org/3/library/sys.html)

Status de saída

- A `sys` biblioteca também possui métodos integrados. Podemos usá-los `sys.exit(i)` para encerrar o programa com um código de saída específico:

```
# Exits with explicit value, importing sys

import sys

if len(sys.argv) != 2:
    print("Missing command-line argument")
    sys.exit(1)

print(f"hello, {sys.argv[1]}")
sys.exit(0)
```

Observe que a notação de ponto é usada para utilizar as funções internas de `sys`.

Arquivos CSV

- O Python também possui suporte integrado para arquivos CSV.
- Modifique seu código da `phonebook.py` seguinte forma:

```
import csv

file = open("phonebook.csv", "a")

name = input("Name: ")
number = input("Number: ")

writer = csv.writer(file)
writer.writerow([name,number])

file.close()
```

  O Notice `writerow` adiciona as vírgulas no arquivo CSV para nós.

- Embora `file.close` `and` e `or` `file = open` sejam sintaxes comuns e disponíveis em Python, este código pode ser melhorado da seguinte forma:

```
import csv

name = input("Name: ")
number = input("Number: ")

with open("phonebook.csv", "a") as file:

    writer = csv.writer(file)
    writer.writerow([name,number])
```

  Observe que o código está indentado abaixo da `with` instrução. Isso fecha o arquivo automaticamente ao término da execução.

- Da mesma forma, podemos escrever um dicionário da seguinte maneira dentro do arquivo CSV:

```
import csv

name = input("Name: ")
number = input("Number: ")

with open("phonebook.csv", "a") as file:

    writer = csv.DictWriter(file, fieldnames=["name", "number"])
    writer.writerow({"name": name, "number": number})
```

  Observe que este código é bastante semelhante à nossa iteração anterior, mas com `csv.DictWriter` em vez de.

Bibliotecas de terceiros

- Uma das vantagens do Python é sua enorme base de usuários e o número igualmente grande de bibliotecas de terceiros.
- Você pode instalar a biblioteca CS50 em seu próprio computador digitando o comando `pip install cs50`, desde que tenha o Python (https://python.org) instalado.
- Considerando outras bibliotecas, David demonstrou o uso de `cowsay` e `qrcode`.

Resumindo

Nesta lição, você aprendeu como os fundamentos da programação, abordados em lições anteriores, podem ser implementados em Python. Além disso, você aprendeu como o Python permite um código mais simplificado. Você também aprendeu a utilizar diversas bibliotecas Python. Ao final, você aprendeu que suas habilidades como programador não se limitam a uma única linguagem de programação. Você já está percebendo como está descobrindo uma nova forma de aprendizado por meio deste curso, que pode ser útil em qualquer linguagem de programação – e, talvez, em praticamente qualquer área de aprendizado! Especificamente, discutimos…

- Python
- Variáveis
- Condicionais
- Laços
- Tipos
- Programação Orientada a Objetos
- Truncamento e imprecisão de ponto flutuante
- Exceções
- Dicionários
- Argumentos da linha de comando
- Bibliotecas de terceiros

Até a próxima!