Introduction to Python

DCLDP Summer Camp 2023

Christiana Chamon Garcia, Ph.D, EIT

Outline

- How programs work
- Comments/whitespace
- print
- Variables
- User input
- Binary
- Arithmetic
- Boolean
- Functions
- Conditionals
- Loops
- Arrays

How does a program work?

Comments

How to comment

- # single-line comment
- """ multi-line comment """

Whitespace

Whitespace

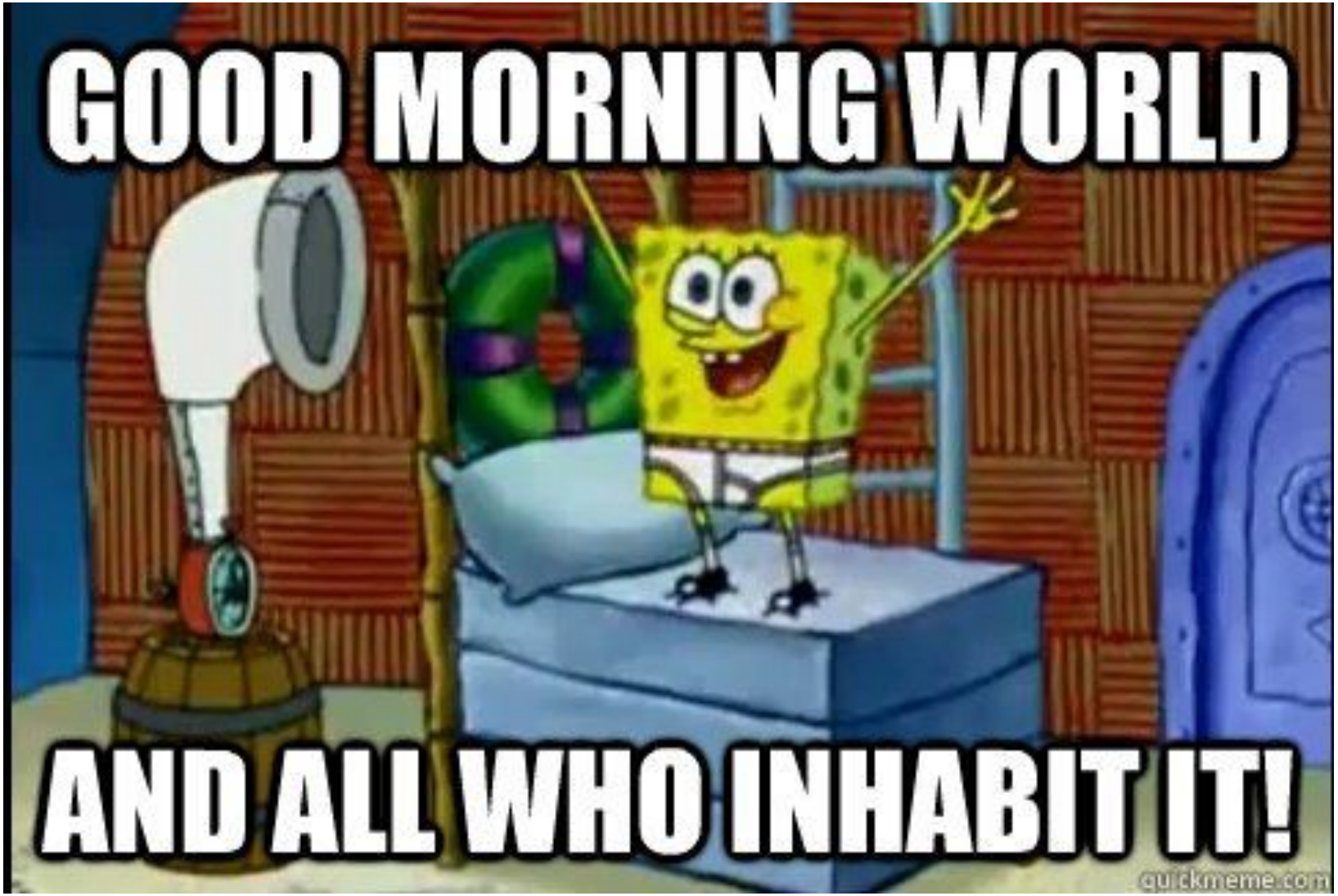
- Indentation is required for functions, loops, conditionals, and other blocks.
- Empty lines are not required, but they are highly useful for separating sections of code.

Python example with whitespace

```
#this example contains whitespace

print("Good morning world")

print("and all who inhabit it!")
```

C example without whitespace

```
#this example doesn’t contain whitespace
print("Good morning world")
print("and all who inhabit it!")
```


GOOD MORNING WORLD
AND ALL WHO INHABIT IT!
quickmeme.com

print

print

- The print() function prints a specified value at the output window.
- The value can be any literally anything as long as it follows valid Python syntax.

Examples

- print(“John is awesome”) //valid
  - “John is awesome” is a string
- print(1) //valid
- print(x);
  - Is x a defined int variable? If so, it’s valid; otherwise, it’s invalid.
- print(‘a’); //valid
  - ‘a’ is a character
- print(“I can’t escape”) //valid
- print(“I\’ve escaped”) //valid

String syntax

- In the character example, we used apostrophes.
- In the string example, we used quotation marks.
- Characters can use apostrophes or quotation marks.
- Because we use apostrophes for characters, we typically use an escape sequence “\”

Char vs. string example

```
c = 'c'
print(c)
print("\n") #adding a newline to separate
the two outputs
e = "eat"
print(e)
```

Variables

Bigger picture

Identifiers

- Variable names can start with letters.
- Variable names can contain letters, numbers, and underscores.
- Variable names cannot start with numbers or punctuation.
- Variable names cannot be built-in functions.
- The camel method is the best practice for naming variables.

Examples

- DCLDP_2023 //valid
- 101programming //invalid
- drChamon //valid, camel
- _camprulez //valid
- int //invalid

Assignments

- Values can be assigned to variables.
- Variables must always be on the left side.
- Values must always be on the right side.

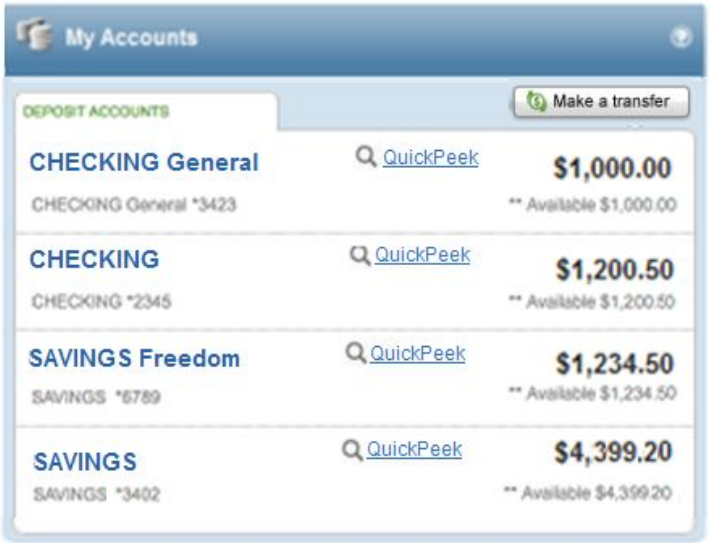
Examples

- x = 5
- y = 7
- z = x + y

Examples in Python

```
x = 5
y = 7
z = x+y
print(x,y,z)
```

Analogy time!!!



type() function

- The type() function tells us the type of a variable.
- In Python, any time a variable is assigned, its type is automatically assigned.

User Input

User input

- As shown, we can output (print) the contents of a variable to the window.
- We can also input data to a variable.
- The input() function allows the user to store data into a variable mid-program.

User input example

```
print("Enter a number: ")
numin = input()
print("Your number is: ", numin)
print("Enter a letter, word, or sentence: ")
stringin = input()
print("Your string is: ", stringin)
```

Binary

Base 2

- Most humans speak in Base 10.
- Computers speak in Base 2, or binary.
- Any command you give to your computer will be translated into binary.
- Binary strings are made up of bits.

Some binary examples

- 2_2: 10
- 4_2: 100
- 5_2: 101
- 7_2: 111
- 42_2: 101010
- 64_2: 1000000

Two's Complement

- Converting binary numbers to their negative representation is easy as 1-2-3
  1. Flip all the bits (1s become 0s and vice versa)
  2. Add 1
  3. Pat yourself on the back
- Example: 101010 (42) becomes 010110 (-42)

Two's Complement

- Here's some supplementary information.
- When in doubt, ask a TA! :)

Unsigned vs. signed bits

- Here's some supplementary information.
- When in doubt, ask a TA! :)

In-class activity

- Let's convert a few numbers to binary and a few binary numbers back to Base 10!
- We'll use the mod-2 method to convert to binary.
- We'll use the powers-of-2 and double-dabble methods to convert to Base 10.
- We'll then take its negative and use Two's Complement to represent it in binary.

In-class challenge

1. Pick a number (Base 10)
2. Convert it to binary
3. Using any method you'd like, convert it back to Base 10
4. Use Two's Complement to give the negative binary representation.

Arithmetic

Standard operators

- Addition: +
- Subtraction: -
- Multiplication: *
- Division: /
- Modulo: %

Order of operations

1. ()
2. *, /, or % from left to right
3. + or - from left to right

Try this code!

```
x = (4 + 5)
y = 2 * 4
z = x % y
print(z)
```

Bitwise operators

- AND: &
- OR: |
- XOR: ^
- NOT: ~
- Shift-left: <<
- Shift-right: >>

NOTE: bitwise operators can only be used on char and int variables.

Bitwise XOR

- In Python, we have a bitwise XOR: **^**
- Python does not have a built-in logical XOR
  - A function can be created
- Alternatively, the not-equal logical operator **!=**.

Bitwise shifting application

- If the data from a file is a 16b number, but the transfer protocol works only in 8b, we use bitwise shifting to combine the data.
- This is a quick-and-easy fix.

Try this code!

```
a = 4        #100
b = 3        #011
c = a & b  #play with bitwise operators
print(c)
```

More on bitwise operators

- Bitwise operators work directly with binary numbers.
- In order to understand bitwise operation, one must understand binary numbers.
- Conversion from decimal to binary can be done via modulus-2 and divide-by-2.
- Conversion from binary to decimal can be done via adding powers of 2 or double-dabble.

In-class activity

- Challenge: what will be the resultant of each bitwise operation?
- Reminder: !0=1, 1&0=0, 1|0=1, 1^1=0
- Hint: convert all of the values to binary first!
- Please show your process.

1. !42
2. 4&3
3. 5|2
4. 7^4
5. 5<<1
6. 64>>3

Boolean

Values

- true (logical 1)
- false (logical 0)

Logical Operators

- Greater than: >
- Less than: <
- Equal to: ==
- Not equal to: !=
- Greater than or equal to: >=
- Less than or equal to: <=
- AND: &&
- OR: ||
- NOT: !

Try this code!

```
a = 4
b = 3
c = a > b
print(c)
```

Functions

Function overview

- The purpose of a function is to replace the act of repeating the same lines.
- Instead of copying and pasting the same chunk of code for different sequences, the user can define a function and use it whenever needed.
- Functions make it easier for the user to debug!

Example

Function: switching lanes

1. Raise/lower the turning signal lever
2. Check to see if the path is clear
3. If it's clear, angle the steering wheel
4. Once you've switched lanes, straighten the steering wheel

Example

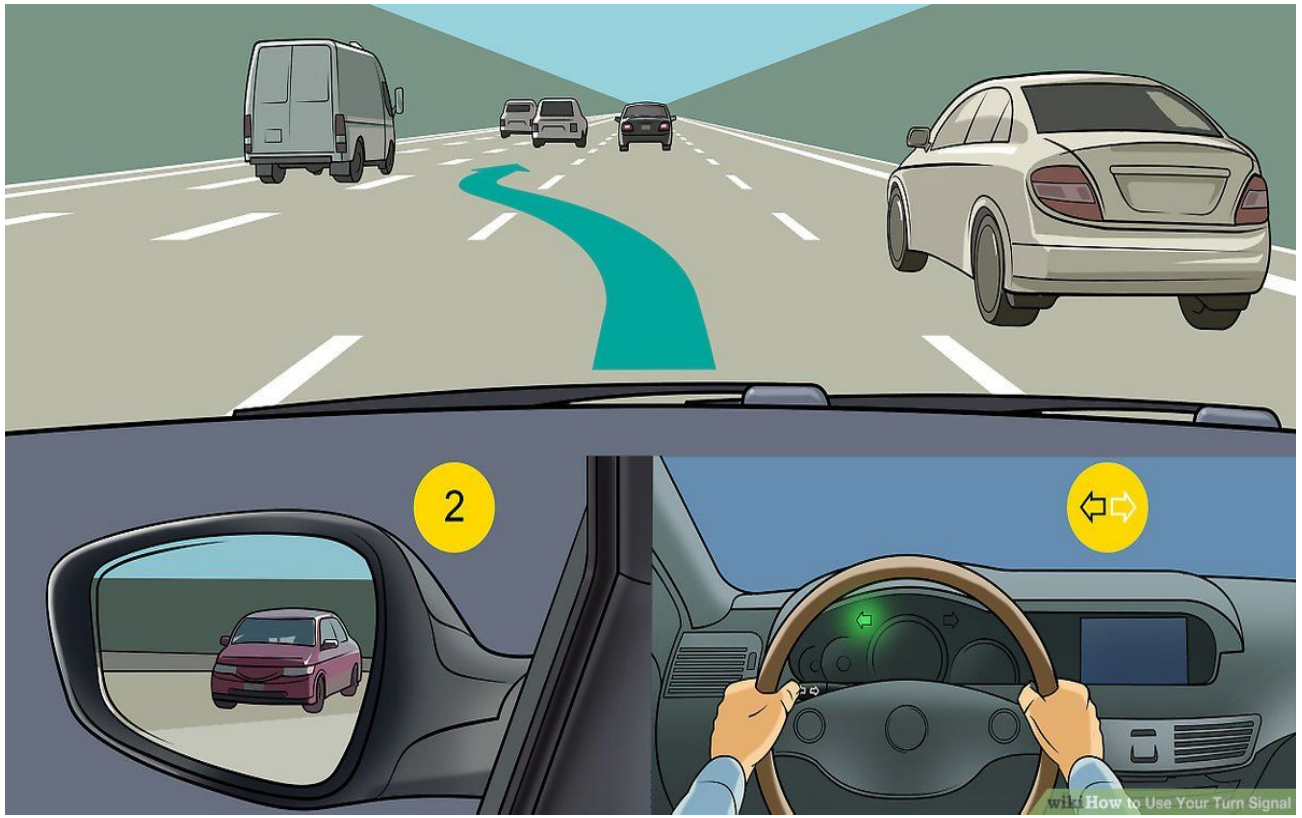


IMAGE LINK

Example

Function: turn 90 degrees (clockwise)

1. pivot your right foot to point to the right of you
2. pick up your left foot
3. turn your body in the direction of which your right foot is pointing
4. put down your left foot

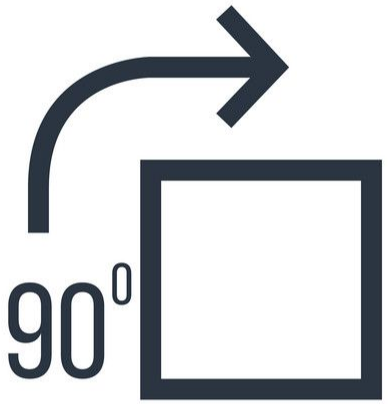


IMAGE LINK

Verdict

- Making those examples into a single function and just referencing them when you need them saves you many lines of code.
- It is ill-advised to write a program that doesn't **function**! :)

Built-In Functions

Examples

- ceil() rounds **up**
- floor() rounds **down**
- sqrt() returns the square root
- pow() returns the power
- abs() returns the absolute value
- randint() generates a random number

NOTE: almost all of these functions only work if you import the appropriate library (i.e. math, random).

Try this code!

```
import math
import random

a = 1.5
b = 2.7
c = 9

x = math.ceil(a)
y = math.floor(b)
z = math.sqrt(c)

print(a,x)
print(b,y)
print(c,z)

q = math.pow(z,2)
print(z,q)

r = -5
s = abs(r)
print(r,s)

min_val = 1
max_val = 100
t = random.randint(min_val,max_val)
print(t)
```

User-Defined Functions

Product of two numbers

```
def productNumbers(a, b):
    product = a*b
    return product

x = 3
y = 4
z = productNumbers(x,y)
print("product = ", z)
```

Adding 5

```
def add5(a):
    return a+5


x = 3
z = add5(x)
print(z)
```

Multiplying by 10

```
def x10(a):
    return a*10

x = 10
print(x10(x))
```

Alphabet ranking

```
def arank(c1, c2):
    return c1>c2

a1 = 'a'
a2 = 'b'
print(arank(a1,a2))
```

Add “ and throw it all away”

```
def tiaa(x):
    return x + " and throw it all away"

a1 = "Do your homework"
print(tiaa(a1))
```

IMAGE LINK

Convert decimal to binary

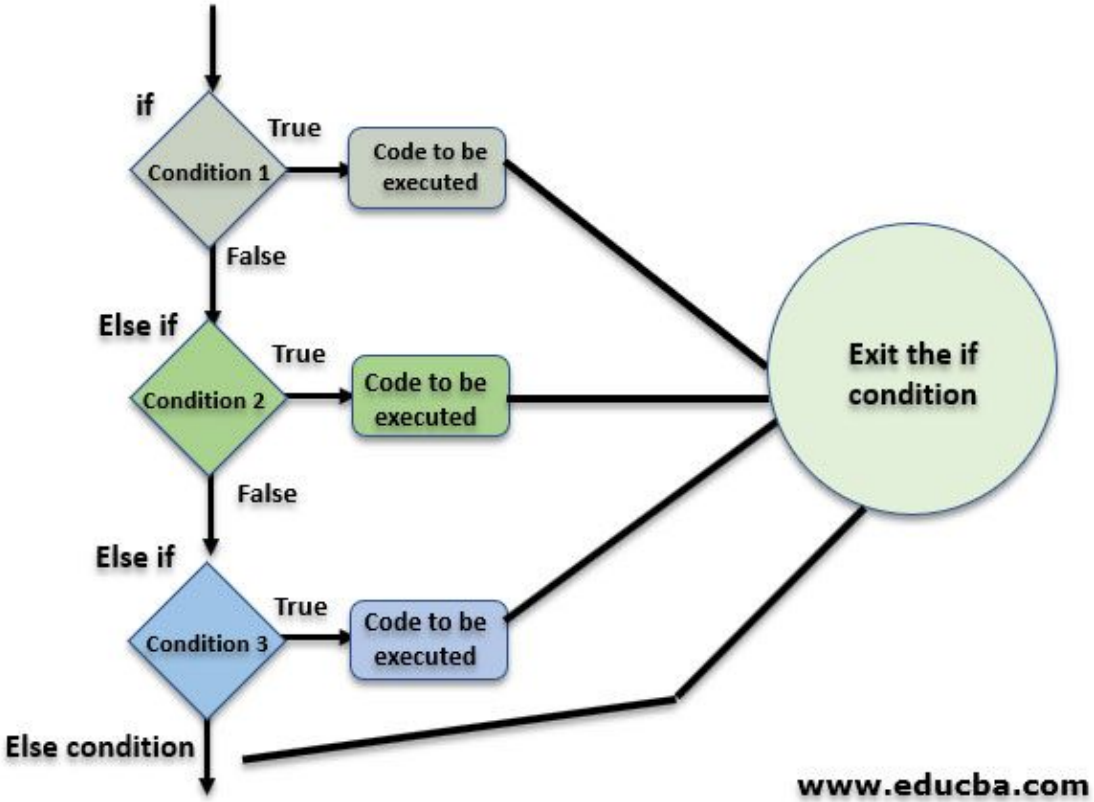
```
import math

def d2b(dec):
    bin = 0
    mod = 0
    temp = 1

    while (dec!=0):
        mod = int(dec)%2
        dec /= 2
        bin = bin + mod*temp
        temp *= 10
    return bin

decimalnum = 42
print(d2b(decimalnum))
```

In-class Activity

- Write a function that takes in two integers as arguments and returns the quotient
- Additional challenge: use two numbers generated by randint()

Conditionals

if, else if, else



LINK TO IMAGE

Try this code!

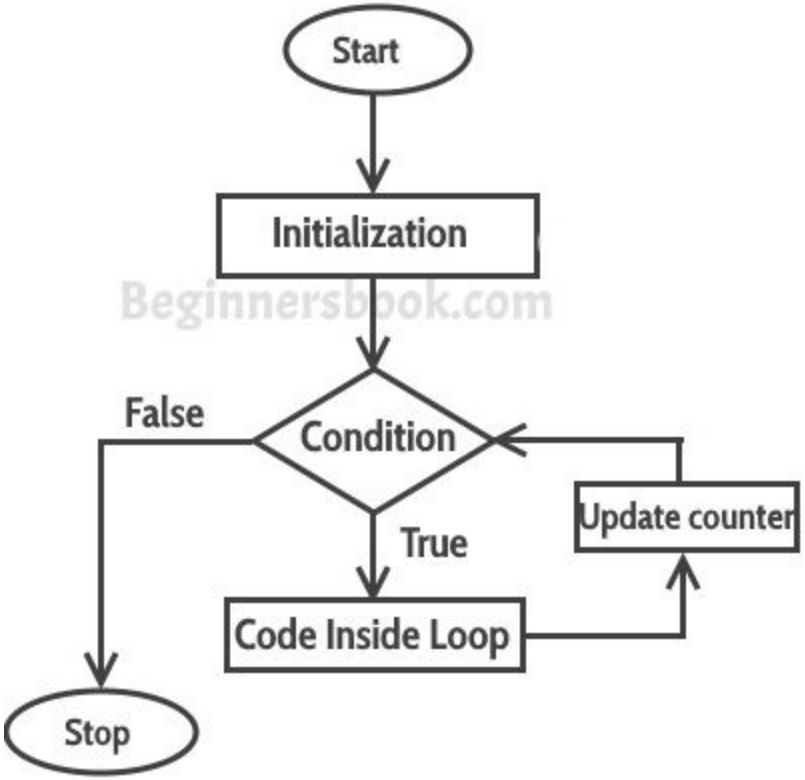
```
a = 4
b = 3

if (a > b):
    print("This is the output of the if-statement")
elif (a < b):
    printf("This is the output of the else if-statement")
else:
    printf("This is the output of the else statement")
```

Activity

- Create a grading system using conditionals (e.g. if >90, A).
- At the output, show the grade and the associated letter.

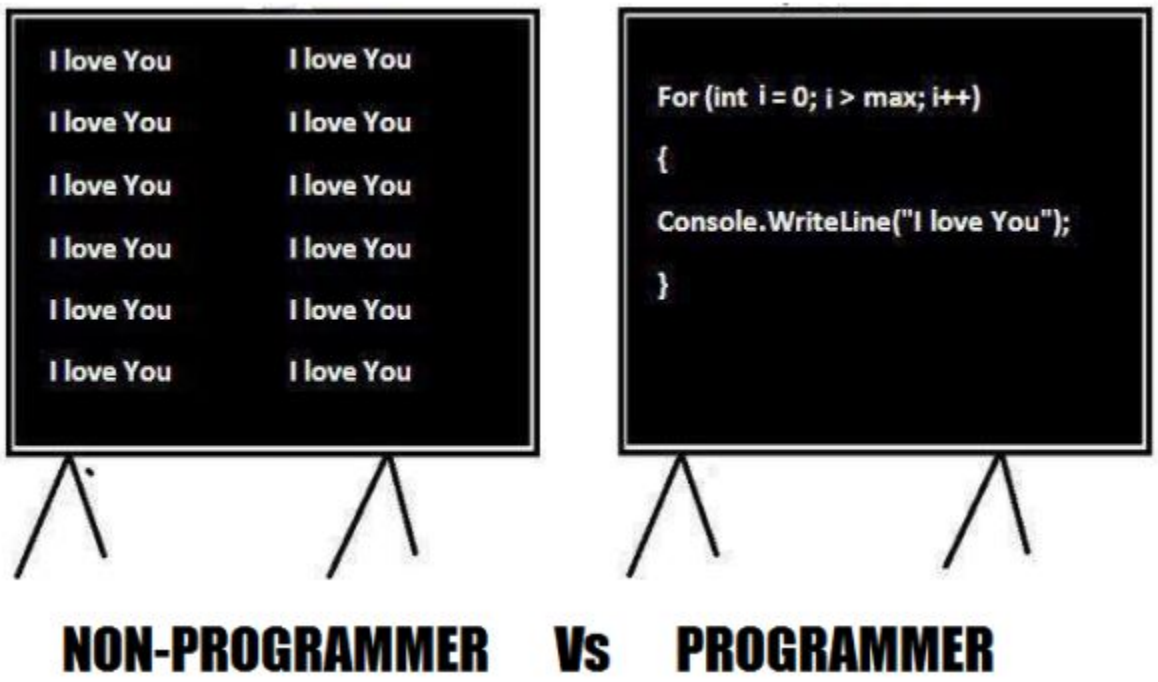
Loops

How loops work



LINK TO IMAGE

Loop purpose



LINK TO IMAGE

Principles of loops

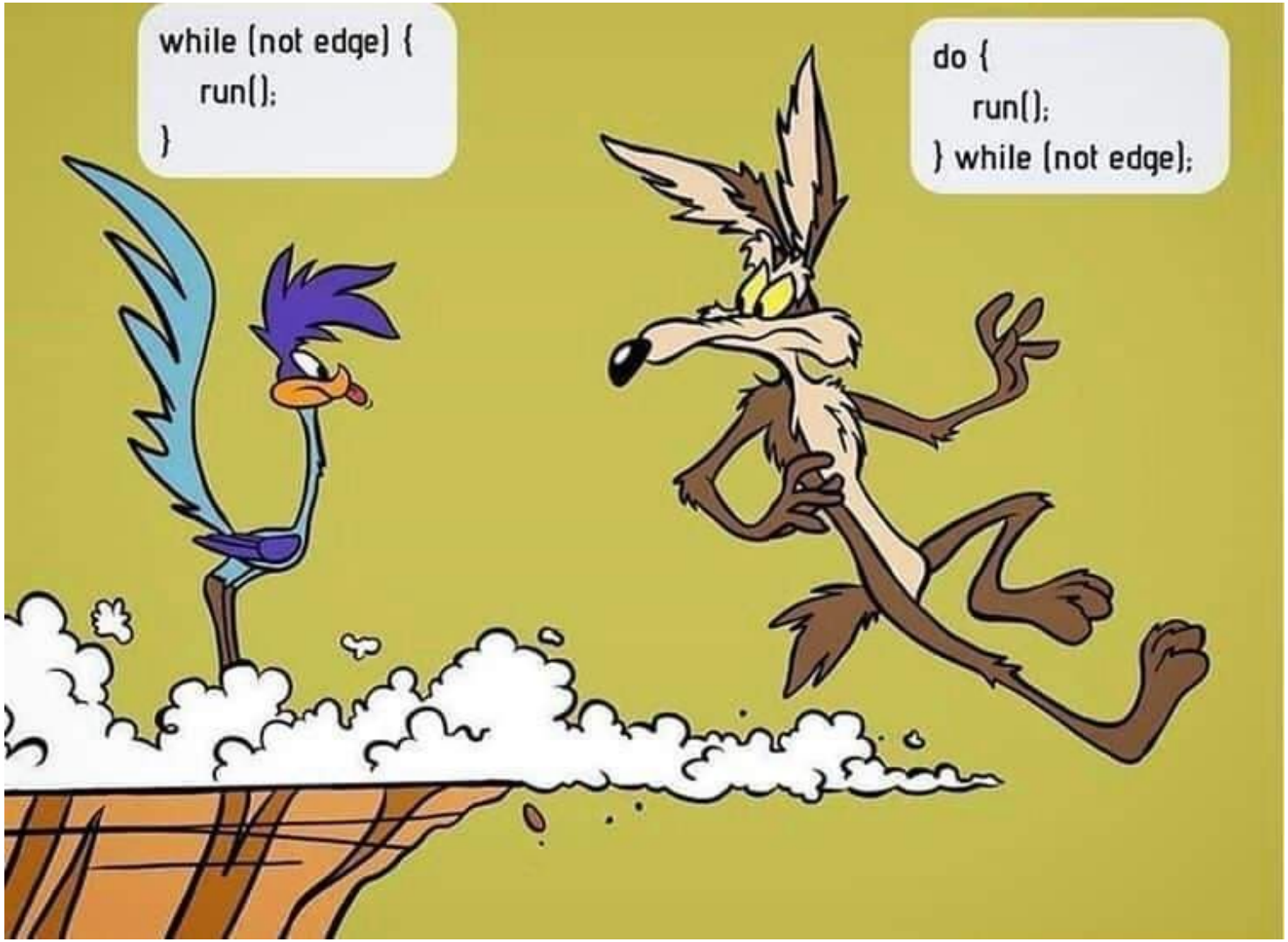
1. Initialization (refer to **variables** section)
2. Conditional (refer to **conditionals** section)

Loop Types

Loop types

- for loops: “For this initialization, keep doing this as long as this condition is met.”
- while loops: “While this condition is met, keep doing this.”

NOTE: Python doesn’t have an inherent “do while” loop.



LINK TO IMAGE

LINK TO IMAGE

Try this code!

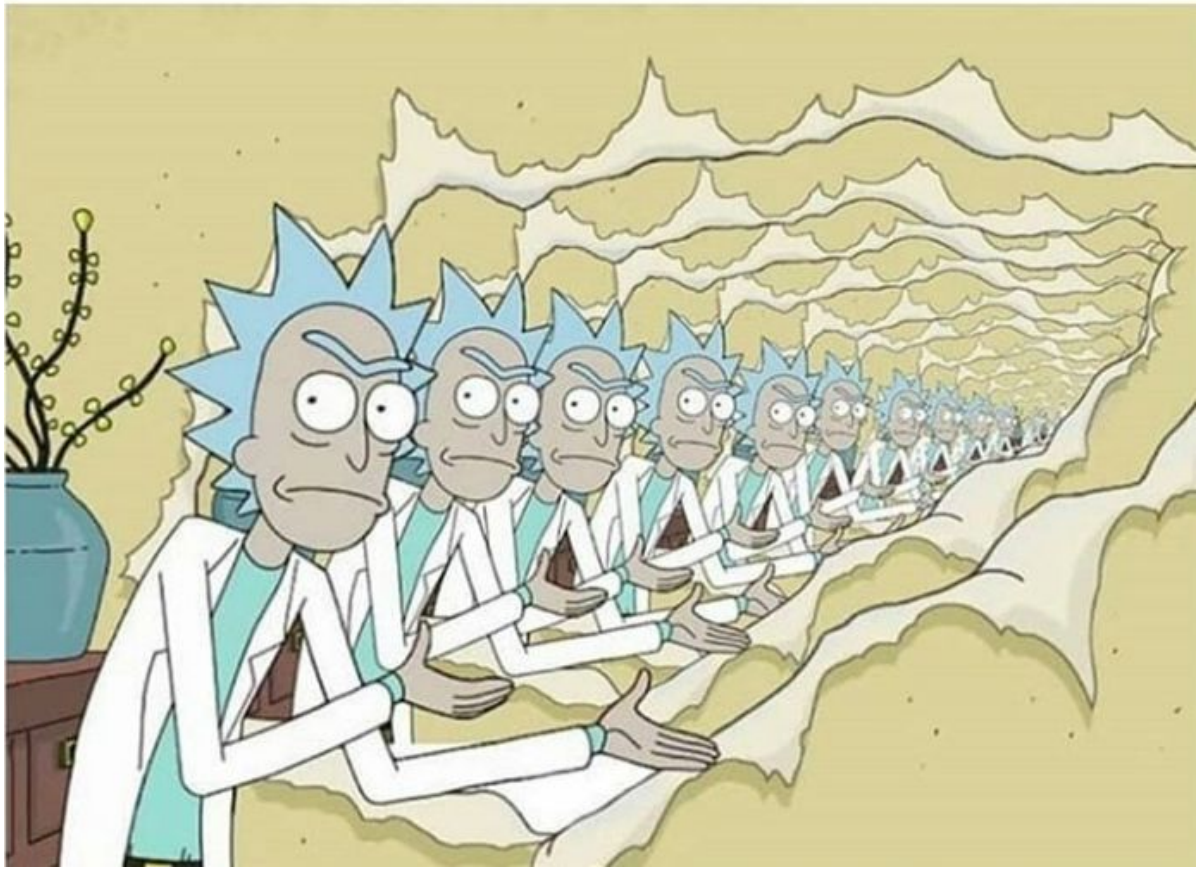
```
for a in range(2):
    print(a);

print("\n")

b = 2
while (b>0):
    print(b)
    b-=1
```

Exit Statements

break and continue statements

- The break statement takes the user out of the loop.
- The continue statement takes the user to top of the loop, ignoring what's below it.

When you forget to break out of the while loop



LINK TO IMAGE

Try this code!

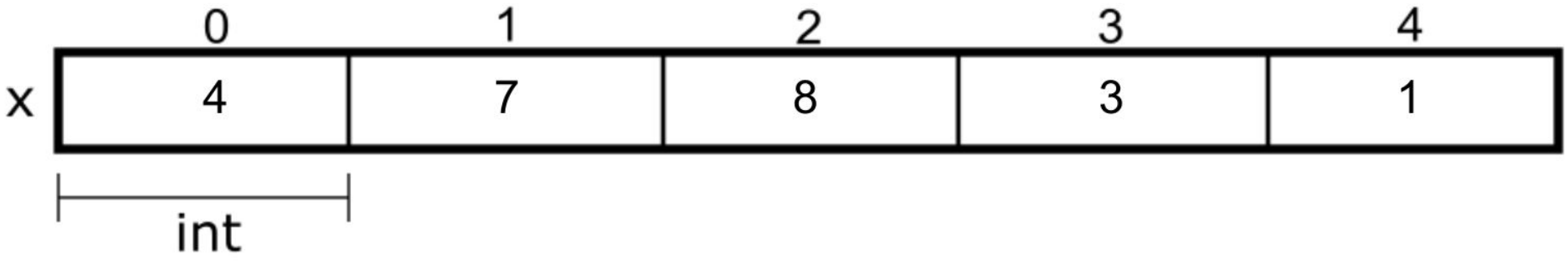
```
for a in range(5):
    if (a==1):
        continue
    elif (a==4):
        break
    print(a)
```

Arrays

Array basics

- The Oxford dictionary defines an array as an impressive display or range of a particular type of thing.
- An array can be of any data type and have as many elements as the user wishes to define.
- 1D arrays are also called vectors.

Example: x = [4,7,8,3,1]

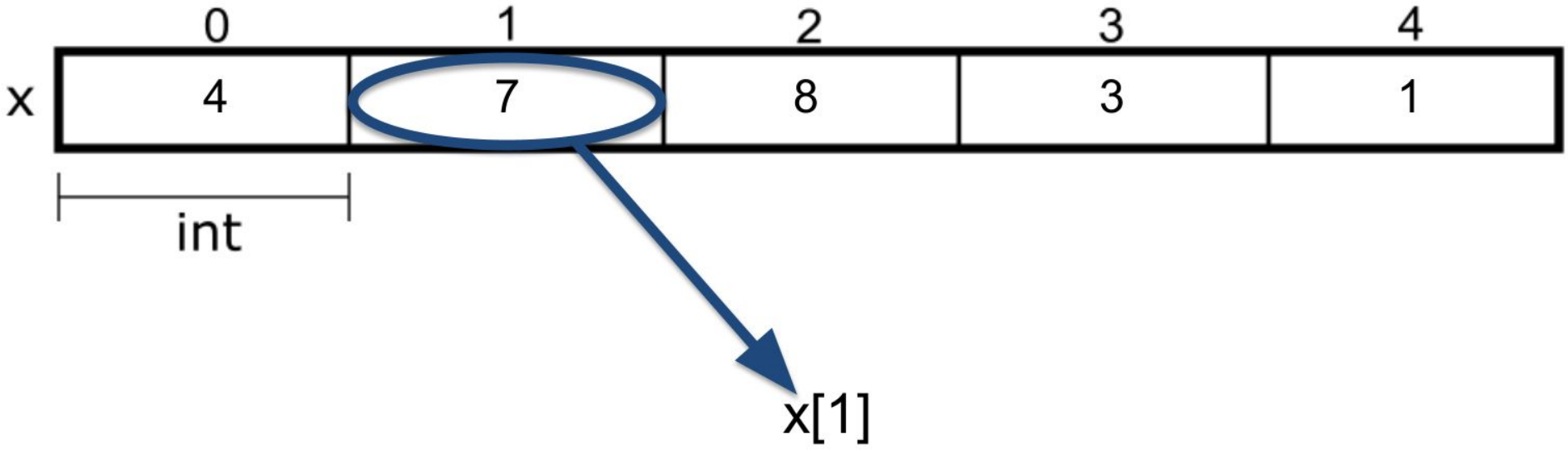
- 

Try this code!

```
x = [4,7,8,3,1]
for i in range(len(x)):
    print(i,x[i])
```

Array Manipulation

Referencing elements

- 

Try this code!

```
x = [4,7,8,3,1]
for i in range(len(x)):
    print(i,x[i])

refnum = 2
print(refnum, x[refnum])
```

Modifying elements

- A user can also modify an element in a vector, replacing the previous element with a new element.
- One way to accomplish element modification is to reference the vector index and assign it a new value.

Try this code!

```
print("Old array")
x = [4,7,8,3,1]
for i in range(len(x)):
    print(i,x[i])
print("\n")
refnum = 2
x[refnum] = 5
print("New array")
for i in range(len(x)):
    print(i,x[i])
```

Strings

Character arrays

- In programming, a string is an array of characters.
- Words are examples of strings, and their letters are examples of characters.
- This sentence is also an example of a string.
- In Python, you can select the range and occurrence by adding [min:max:occurrence] at the end of the variable.
  - Not specifying a min/max assumes beginning/end
  - Not specifying an occurrence assumes “every”

Try this code!

```
x = "Hello World"
for i in range(len(x)):
    print(i,x[i])
```

Try this code!

```
x = "Hello World"
print(x[2:5:1]) #play with this
```

Reversing a string

```
x = "Hello World"
print(x)
print(x [::-1])
```

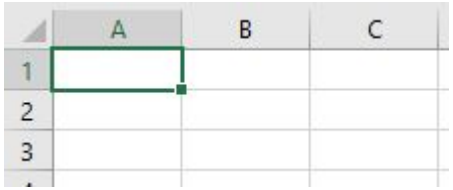
In-class activity

- Write a Python program to print every other letter in a string.
- Hint: use [::2]
- Additional challenge: put it in a function, with the string as the argument.

Matrices

IMAGE LINK

2D arrays



- A matrix is a 2D array.
- Spreadsheets and tables are examples of matrices.

Eve's probing bit	M	CCC_u	p_u	σ_u	CCC_i	p_i	σ_i	CCC_p	p_p	σ_p
HH	0	0.21314			0.67455			0.28482		
LL		0.67377			0.21317			0.28502		
HL		-0.00118			0.00128			-0.00126		
LH		1	1	0	1	1	0	1	1	0
HH	0.1	0.21087			0.67048			0.28198		
LL		0.67090			0.21326			0.28573		
HL		-0.00030			-0.00007			0.00095		
LH		0.99504	1	0	0.99505	1	0	0.99005	1	0

IMAGE LINK

2D arrays

- If you know how to create, access, and traverse (CAT) 1D arrays, you know how to CAT arrays of any dimension.
- CAT 2D arrays (matrices) is nothing more than CAT 1D arrays with an added index.
- CAT 3D arrays is nothing more than CAT 1D arrays with two added indices.

Matrix initialization

```
a = [[1,2],[3,4]]
print(a[0][0]) #play with this
```

Activity: tabulating grades!

- Write a Python program that displays a list of assignment names and their grades.
- Additional challenge: calculate the averages of each type of assignment.

NOTE: in Python, arrays aren't required to be of a homogeneous data type.

Sample code

```
assignments =
[["Hw1",95],["Hw2",85],["Hw3",90]]
print(assignments[1]) #will print just the
one row
```

“If you find yourself in a hole, the first thing to do is stop digging.” ~Texas Bix Bender