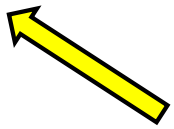
DSCI 510
PRINCIPLES OF PROGRAMMING FOR DATA SCIENCE

Itay Hen

Python Terminal

```
Last login: Sun Aug 17 09:40:52 on ttys005
itayhen@Mac ~ % python
zsh: command not found: python
itayhen@Mac ~ % python3
Python 3.13.4 (v3.13.4:8a526ec7cbe, Jun  3 2025, 21:14:54) [Clang 16.0.0 (clang-
1600.0.26.6)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```



What next?

Python Terminal

```
Last login: Sun Aug 17 09:40:52 on ttys005
itayhen@Mac ~ % python
zsh: command not found: python
itayhen@Mac ~ % python3
Python 3.13.4 (v3.13.4:8a526ec7cbe, Jun  3 2025, 21:14:54) [Clang 16.0.0 (clang-
1600.0.26.6)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> x=1
>>> print(x)
1
>>> x=x+1
>>> print(x)
2
>>> exit()
itayhen@Mac ~ %
```

This is a good test to make sure that you have Python correctly installed. Note that `quit()` also works to end the interactive session.

True or False?

- Easy to read, learn and write — **True**
- Fast program development — **True**
- Fast runtime and memory efficient — **False**
- Vast library support — **True**
- Free and open source — **True**
- Excellent for mobile platforms — **False**
- Most in-demand language 2022 — **True**

Python is Popular!

Jan 2022	Jan 2021	Change	Programming Language	Ratings	Change
1	3	↑	Python	13.58%	+1.86%
2	1	↓	C	12.44%	-4.94%
3	2	↓	Java	10.66%	-1.30%
4	4		C++	8.29%	+0.73%
5	5		C#	5.68%	+1.73%
6	6		Visual Basic	4.74%	+0.90%
7	7		JavaScript	2.09%	-0.11%
8	11	↑	Assembly language	1.85%	+0.21%
9	12	↑	SQL	1.80%	+0.19%
10	13	↑	Swift	1.41%	-0.02%
11	8	↓	PHP	1.40%	-0.60%
12	9	↓	R	1.25%	-0.65%
13	14	↑	Go	1.04%	-0.37%
14	19	⇈	Delphi/Object Pascal	0.99%	+0.20%
15	20	⇈	Classic Visual Basic	0.98%	+0.19%
16	16		MATLAB	0.96%	-0.19%

- Driven by:
  - Ease of programming
  - Data Science/Machine Learning Libraries
  - Community
- Most popular programming languages: 1965 – 2019
  - https://www.youtube.com/watch?v=Og847HVwRSI&t=9s

Elements of Python

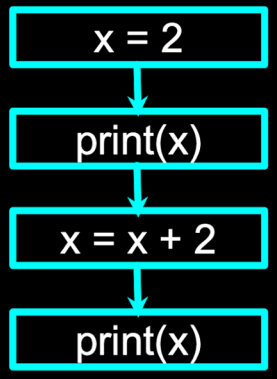
- **Vocabulary / Words** – Vocabulary and Reserved words (Chapter 2)
- **Sentence structure** – valid syntax patterns (Chapters 3-5)
- **Story structure** – constructing a program for a purpose

Program Steps or Program Flow

- Like a recipe or installation instructions, a program is a **sequence** of steps to be done in order.
- Some steps are **conditional** – they may be skipped.
- Sometimes a step of a group of steps is to be **repeated**.
- Sometimes we store a set of steps to be used over and over as needed several places throughout the program (Chapter 4)

Sequential Steps

Sequential Steps



Program:

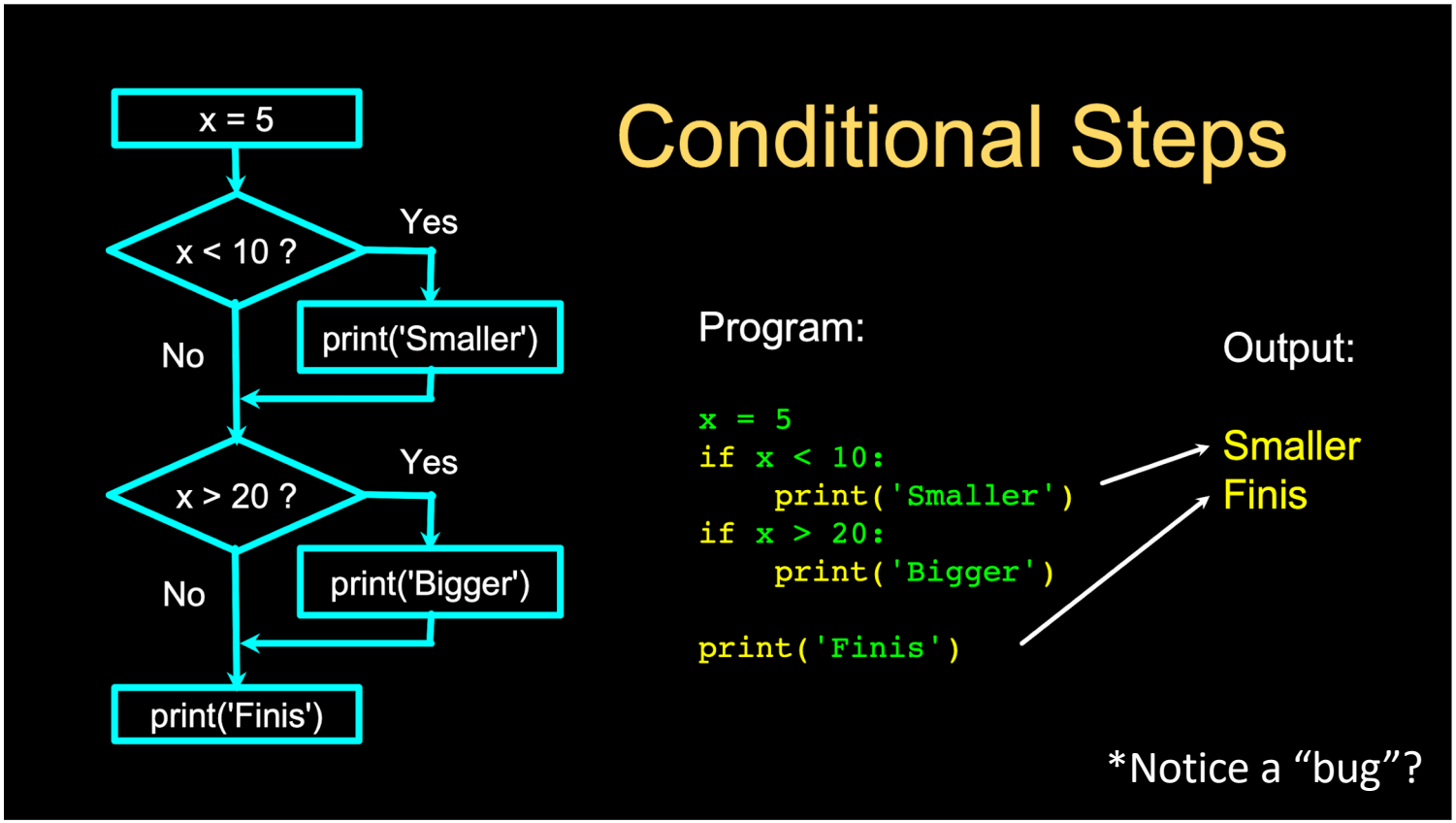
```
x = 2
print(x)
x = x + 2
print(x)
```

Output:

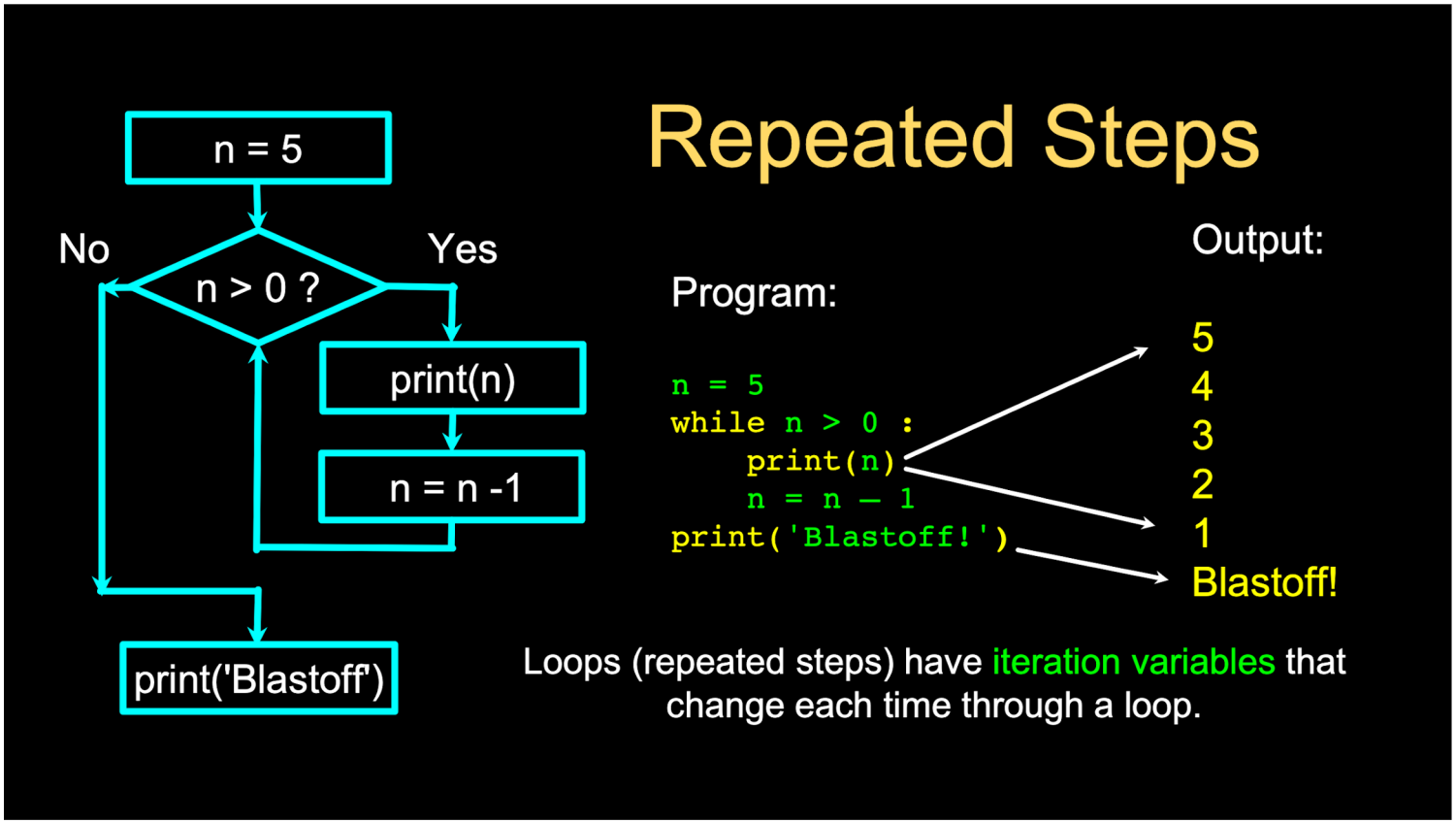
```
2
4
```

When a program is running, it flows from one step to the next. As programmers, we set up "paths" for the program to follow.

Conditional Steps



Repeated Steps



Python Program Flow

```
name = input('Enter file:')
handle = open(name, 'r')

counts = dict()
for line in handle:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word,0) + 1

bigcount = None
bigword = None
for word,count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

print(bigword, bigcount)
```

Sequential

Repeated

Conditional

Python Program: Count Words in a File

```
name = input('Enter file:')
handle = open(name, 'r')

counts = dict()
for line in handle:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word,0) + 1

bigcount = None
bigword = None
for word,count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

print(bigword, bigcount)
```

A short Python "Story" about how to count words in a file

A word used to read data from a user

A sentence about updating one of the many counts

A paragraph about how to find the largest item in a list

Python Scripts

- Interactive Python is good for experiments and programs of 3-4 lines long.
- Most programs are much longer, so we type them into a file and tell Python to run the commands in the file.
- In a sense, we are "giving Python a script".
- As a convention, we add ".py" as the suffix on the end of these files to indicate they contain Python.

Interactive versus Script

- Interactive
  - You type directly to Python one line at a time, and it responds
- Script
  - You enter a sequence of statements (lines) into a file using a text editor and tell Python to execute the statements in the file.

What can go Wrong?

- **Syntax errors**: You have violated the "grammar" rules of Python
- **Logic errors**: The program has good syntax, but there is a mistake in the order of the statements or perhaps a mistake in how the statements relate to one another
- **Semantic errors**: The program is perfectly correct, but it does not do what you intended it to do
- **Runtime errors:** errors that occur while a program is running, typically due to invalid operations like dividing by zero, accessing out-of-bounds memory, or using unavailable resources.

What could possibly go wrong?

Consider this Python program:

```
if 1 + 2 = 3
    print "true"
```

Do you see anything wrong?

What could possibly go wrong?

Bad Python program

```
if 1 + 2 = 3
   print “true”
```

Correct Python Program

```
if 1 + 2 == 3:
    print("true")
```

- Equality operator is double equal (==) Single equal (=) is used for assignments
- Missing colon (:) at the end of condition
- Missing parentheses for print() function
- Quotation marks must be plain ASCII, not left/right quotation mark
- Indentation should be in multiples of 4 (not 3) recommended

PYTHON CONSTANTS AND VARIABLES

Python – Variables and Statements

- Variables
  - Weakly typed
  - Double-edged sword
  - Reserved words/keywords
  - Valid names
- Statements
  - Statements are where [small] things get done
  - Often also referred to as “lines” of code

Constants

Constants

- Fixed values such as numbers, letters, and strings, are called "constants" because their value does not change
- Numeric constants are as you expect
- String constants use single quotes (') or double quotes (")

```
>>> print(123)
123
>>> print(98.6)
98.6
>>> print('Hello world')
Hello world
```

*Unlike other languages, Python doesn't really support built-in, enforced constants

Reserved Words

You cannot use reserved words as variable names / identifiers

```
False   class   return  is      finally
None    if      for     lambda  continue
True    def     from    while   nonlocal
and     del     global  not     with
as      elif    try     or      yield
assert  else    import  pass
break   except  in      raise
```

(Your code editor will highlight reserved words)

Built-in functions: https://docs.python.org/3/library/functions.html

Variables

Variables

- A variable is a named place in the memory where a programmer can store data and later retrieve the data using the variable "name"
- Programmers get to choose the names of the variables
- You can change the contents of a variable in a later statement

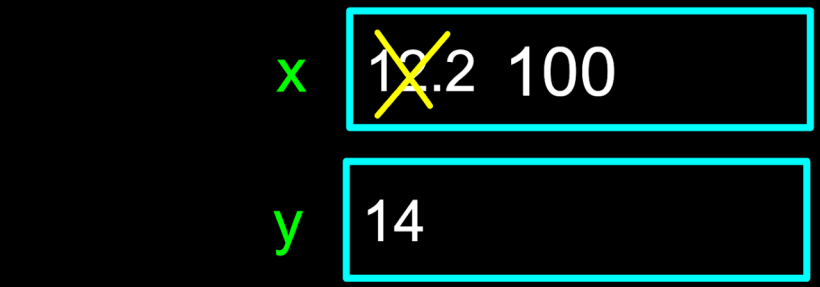
```
x = 12.2
y = 14
```

Variables

Variables

- A variable is a named place in the memory where a programmer can store data and later retrieve the data using the variable "name"
- Programmers get to choose the names of the variables
- You can change the contents of a variable in a later statement

```
x = 12.2
y = 14
x = 100
```



Python Variable Name Rules

- Must start with a letter or underscore _
- Must consist of letters, numbers, and underscores
- Case Sensitive

```
Good:      spam     eggs    spam23     _speed
Bad:       23spam      #sign   var.12
Different:     spam    Spam    SPAM
```

Mnemonic Variable Names

- Since we programmers are given a choice in how we choose our variable names, there is a bit of “best practice”
- We name variables to help us remember what we intend to store in them (“mnemonic” = “memory aid”)
- This can confuse beginning students because well-named variables often “sound” so good that they must be keywords

http://en.wikipedia.org/wiki/Mnemonic

```
x1q3z9ocd = 35.0
x1q3z9afd = 12.50
x1q3p9afd = x1q3z9ocd * x1q3z9afd
print(x1q3p9afd)
```

What is this bit of code doing?

```
x1q3z9ocd = 35.0
x1q3z9afd = 12.50
x1q3p9afd = x1q3z9ocd * x1q3z9afd
print(x1q3p9afd)
```

```
a = 35.0
b = 12.50
c = a * b
print(c)
```

What are these bits of code doing?

```
x1q3z9ocd = 35.0
x1q3z9afd = 12.50
x1q3p9afd = x1q3z9ocd * x1q3z9afd
print(x1q3p9afd)
```

```
a = 35.0
b = 12.50
c = a * b
print(c)
```

What are these bits of code doing?

```
hours = 35.0
rate = 12.50
pay = hours * rate
print(pay)
```

Python Styleguide

- PEP8
  - https://peps.python.org/pep-0008/

PYTHON: EXPRESSIONS, STATEMENTS

Sentences or Lines

```
x = 2
x = x + 2
print(x)
```

← Assignment statement

← Assignment with expression

← Print statement

Variable Operator Constant Function

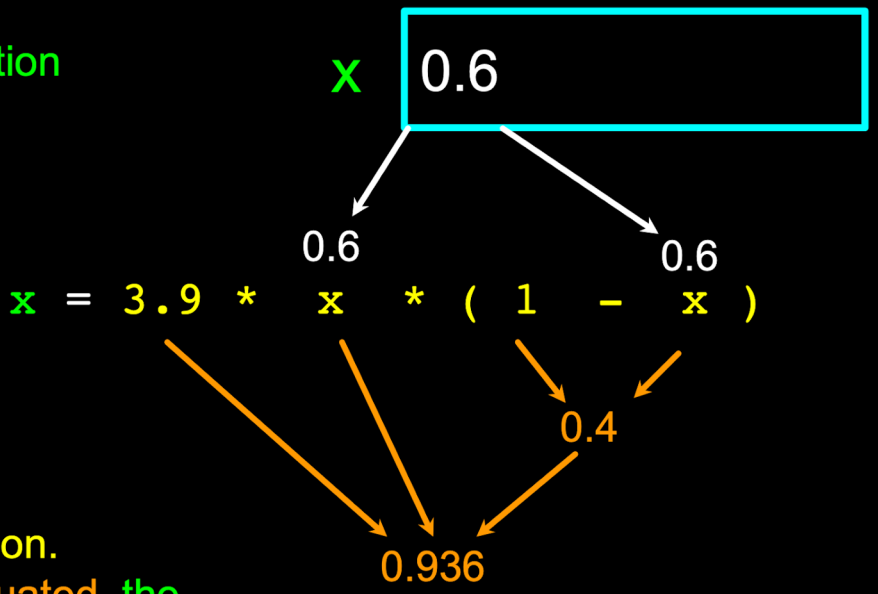
Assignment Statements

- We assign a value to a variable using the assignment statement (=)
- An assignment statement consists of an expression on the right-hand side and a variable to store the result

```
x = 3.9 * x * ( 1 - x )
```

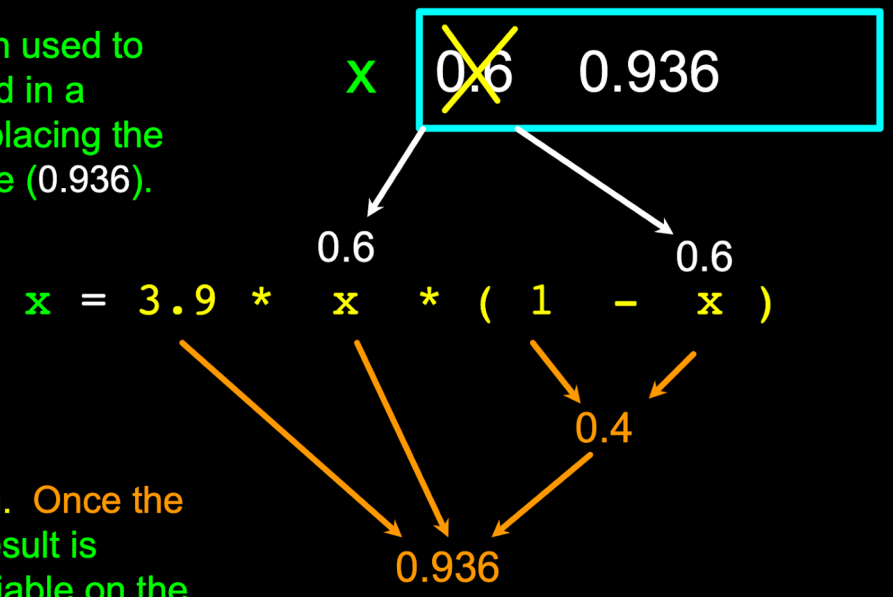
```
x = 0.6
```

A variable is a memory location used to store a value (0.6)



The right side is an expression. Once the expression is evaluated, the result is placed in (assigned to) x.

A variable is a memory location used to store a value. The value stored in a variable can be updated by replacing the old value (0.6) with a new value (0.936).



The right side is an expression. Once the expression is evaluated, the result is placed in (assigned to) the variable on the left side (i.e., x).

Arithmetic Operators

- Exponentiation looks different
  - `3 ** 2` equals 9
- Asterisk is multiplication
- Modulus is the remained after division:
  - `23 % 6` equals 5
  - "23 modulo 6"
  - `2 % 5` equals 2

Operator	Operation
**	Exponentiation
*	Multiplication
/	Division
+	Addition
-	Subtraction
%	Modulus
//	Quotient/floor division

Order of Precedence

- PEMDAS
  - **P**arentheses
  - **E**xponentiation
  - **M**ultiplication
  - **D**ivision

  Equal
  - **A**ddition
  - **S**ubtraction

  Equal
- **Left to Right**

- Which operator takes precedence?

```
1 + 2 * 3 - 4 / 5 ** 6
```

- So, `5 / 5 * 2` is 2, not `0.5`
- When in doubt, just use parentheses!

Numeric Expressions

```
>>> xx = 2
>>> xx = xx + 2
>>> print(xx)
4
>>> yy = 440 * 12
>>> print(yy)
5280
>>> zz = yy / 1000
>>> print(zz)
5.28
```

```
>>> jj = 23
>>> kk = jj % 5
>>> print(kk)
3
>>> print(4 ** 3)
64
```

```
    4 R 3
  ------
5 | 23
    20
  ----
     3
```

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder

Integer Division

Integer division produces a floating point result

```
>>> print(10 / 2)
5.0
>>> print(9 / 2)
4.5
>>> print(99 / 100)
0.99
>>> print(10.0 / 2.0)
5.0
>>> print(99.0 / 100.0)
0.99
```

This was different in Python 2.x

Integer Division With Remainder

```
30 / 8 == 3.75
```
Regular division

```
30 // 8 == 3
```
30 divided by 8 is 3 (plus some remainder)

```
30 % 8 == 6
```
The remainder when dividing 30 by 8

8 * 3 + 6 == 30

% is called the modulo operator ("30 modulo 8")

// is called the floor division operator

```
n = 247                 # Common type of application
if n % 13 == 0:
    print(n, 'is a multiple of 13')
```

What is the value?

Expression	Value
83 / 10	8.3
83 // 10	8
83 % 10	3
100 ** 3	1000000
100 ** 0.5	10.0
4 * 5 ** 2	100
4 / 5 * 2	1.6
4 / (5 * 2)	0.4

Fun expression: 2 ** 1000 (try on your computer)

PYTHON: TYPES

What Does "Type" Mean?

- In Python variables, literals, and constants have a "type"
- Python knows the difference between an integer number and a string
- For example "+" means "addition" if something is a number and "concatenate" if something is a string

```
>>> ddd = 1 + 4
>>> print(ddd)
5
>>> eee = 'hello ' + 'there'
>>> print(eee)
hello there
```

concatenate = put together

Type defines structure of values and allowed operations on them
Polymorphism: when the same operator behaves differently when applied to different types

Type Matters

- Python knows what "type" everything is
- Some operations are prohibited
- You cannot "add 1" to a string
- We can ask Python what type something is by using the type() function

```
>>> eee = 'hello ' + 'there'
>>> eee = eee + 1
Traceback (most recent call last):
File "<stdin>", line 1, in
<module>TypeError: Can't convert
'int' object to str implicitly
>>> type(eee)
<class'str'>
>>> type('hello')
<class'str'>
>>> type(1)
<class'int'>
>>>
```

Several Types of Numbers

- Numbers have two main types
  - Integers are whole numbers: -14, -2, 0, 1, 100, 401233
  - Floating Point Numbers have decimal parts: -2.5 , 0.0, 98.6, 14.0
- There are other number types - they are variations on float and integer

```
>>> xx = 1
>>> type (xx)
<class 'int'>
>>> temp = 98.6
>>> type(temp)
<class'float'>
>>> type(1)
<class 'int'>
>>> type(1.0)
<class'float'>
>>>
```

https://docs.python.org/3/library/stdtypes.html#numeric-types-int-float-complex