**CIS 210**

- Where do I start? A step-by-step guide to computational problem-solving: how to write a good, working Python program

- Python overview – a closer look
  -- primitive elements
  -- combining primitive elements (expressions)
  -- extending the language
  -- variable assignment
  -- user-defined functions
    -- parameters
    -- return

---

**CIS 210: STEP-BY-STEP COMPUTATIONAL PROBLEM SOLVING (HOW TO WRITE A GOOD, WORKING PYTHON PROGRAM)**

**Thinking:**

- problem: carefully review project specification, clarify;
- write simple examples

- algorithm: review, revise, or create algorithm to solve the problem; write pseudocode; check against examples

- program: design (top down), refer again to project specification, determine functions; for each function:
  - write the function header;
  - write a brief description;
  - write type contract;
  - write test examples (reuse examples);
  - write the return statement;
  - check against examples, pseudocode

---

**CIS 210: STEP-BY-STEP COMPUTATIONAL PROBLEM SOLVING (HOW TO WRITE A GOOD, WORKING PYTHON PROGRAM)**

**Coding:**

- program: coding (bottom up)
  - review pseudocode, function outlines, Python toolkit
  - write Python code

- program: testing and debugging
  - test each function using examples of use
  - revise and retest as needed

---

**Days of coding can save hours of planning.**

---

**CIS 210**

For example:

```
pizza compare
```

Write a function, pizza_compare, to determine the cost per square inch of pizza, given the diameter and the cost of the pizza.
Python overview – a closer look

• what are Python’s primitive elements?
• how can we combine the elements?
• how can we create our own elements?

Recall:

- Keywords define the syntax and structure of the Python language, for example:
  - special values (None)
  - special operations (return)
  - flow of control (if, while)
  - special instructions (def)

```python
>>> help()
...
help> keywords
Here is a list of the Python keywords. Enter any keyword to get more help.
...
help>
```

Recall:

- Identifiers are the names given to Python elements like integers, strings, and functions (e.g., testfunc, time, timeLeft, print, len, abs, math.pi)

```python
>>> help(testfunc)
...
```

- Python overview – a closer look
  - mental model of computational processes
  - for reading/writing more advanced code
  - for analyzing (preventing!) errors
Python overview – a closer look

- what are Python’s primitive elements?

There is only one kind of primitive element in Python

<table>
<thead>
<tr>
<th>OBJECT</th>
<th>type – range of values and operations</th>
<th>value(s)</th>
<th>id – memory location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Examples of Python objects:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;&gt;&gt; 99</td>
<td>value</td>
<td>&gt;&gt;&gt; 99.9</td>
<td>99.9</td>
</tr>
<tr>
<td>&gt;&gt;&gt; type(99)</td>
<td>type</td>
<td>&gt;&gt;&gt; type(99)</td>
<td>&lt;class 'int'&gt;</td>
</tr>
<tr>
<td>&gt;&gt;&gt; id(99)</td>
<td>memory loc</td>
<td>&gt;&gt;&gt; id(99.9)</td>
<td>4298470336</td>
</tr>
</tbody>
</table>

- OBJECT
  type
  value(s)
  id (memory location)

Examples of Python objects:

<table>
<thead>
<tr>
<th>Examples of Python objects:</th>
<th>type – range of values and operations</th>
<th>value(s)</th>
<th>id – memory location</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;&gt;&gt; 'hello'</td>
<td>value</td>
<td>&gt;&gt;&gt; True</td>
<td>True</td>
</tr>
<tr>
<td>&gt;&gt;&gt; type('hello')</td>
<td>type</td>
<td>&gt;&gt;&gt; type(True)</td>
<td>&lt;class 'bool'&gt;</td>
</tr>
<tr>
<td>&gt;&gt;&gt; id('hello')</td>
<td>memory loc</td>
<td>&gt;&gt;&gt; id(True)</td>
<td>4297232736</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Examples of Python objects:</th>
<th>type – range of values and operations</th>
<th>value(s)</th>
<th>id – memory location</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;&gt;&gt; 99</td>
<td>value</td>
<td>&gt;&gt;&gt; 99.9</td>
<td>99.9</td>
</tr>
<tr>
<td>&gt;&gt;&gt; type(99)</td>
<td>type</td>
<td>&gt;&gt;&gt; type(99)</td>
<td>&lt;class 'int'&gt;</td>
</tr>
<tr>
<td>&gt;&gt;&gt; id(99)</td>
<td>memory loc</td>
<td>&gt;&gt;&gt; id(99.9)</td>
<td>4298470336</td>
</tr>
</tbody>
</table>
CIS 210

Python overview – a closer look

✓ what are the primitive elements?
  – Objects, with types that determine range of values and operations for that object, and memory locations

• how can we combine elements?
  -- with expressions

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
<th>Notes</th>
<th>Full documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>x + y</td>
<td>sum of x and y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x − y</td>
<td>difference of x and y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x × y</td>
<td>product of x and y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x ÷ y</td>
<td>quotient of x and y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x ** y</td>
<td>floored quotient of x and y</td>
<td>(1)</td>
<td></td>
</tr>
<tr>
<td>x % y</td>
<td>remainder of x ÷ y</td>
<td>(2)</td>
<td></td>
</tr>
<tr>
<td>¬x</td>
<td>x negated</td>
<td></td>
<td></td>
</tr>
<tr>
<td>abs(x)</td>
<td>absolute value or magnitude of x</td>
<td></td>
<td>abs()</td>
</tr>
<tr>
<td>ceil(x)</td>
<td>x converted to integer</td>
<td>(3)</td>
<td>int()</td>
</tr>
<tr>
<td>floor(x)</td>
<td>x converted to floating point</td>
<td>(4)</td>
<td>float()</td>
</tr>
<tr>
<td>complex(x; y)</td>
<td>a complex number with real part x, imaginary part y</td>
<td>(5)</td>
<td>complex()</td>
</tr>
<tr>
<td>copysign(x; y)</td>
<td>conjugate of the complex number c</td>
<td>(6)</td>
<td></td>
</tr>
<tr>
<td>divmod(x; y)</td>
<td>the pair (x // y, x % y)</td>
<td>(2)</td>
<td>divmod()</td>
</tr>
<tr>
<td>pow(x; y)</td>
<td>x to the power y</td>
<td>(1)</td>
<td>pow()</td>
</tr>
<tr>
<td>x ** y</td>
<td>x to the power y</td>
<td>(3)</td>
<td></td>
</tr>
</tbody>
</table>

CIS 210

• how can we combine elements?
  -- expressions

For example,

```
>>> 99 + 100  # >>> 10
199
>>> 99 < 100  # True

```

expressions are combinations of values (objects) and operators, that can be evaluated and return a result

All numeric types (except complex) support the following operations, sorted by ascending priority (all numeric operations have a higher priority than comparison operations):

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
<th>Notes</th>
<th>Full documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>x + y</td>
<td>sum of x and y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x − y</td>
<td>difference of x and y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x × y</td>
<td>product of x and y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x ÷ y</td>
<td>quotient of x and y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x ** y</td>
<td>floored quotient of x and y</td>
<td>(1)</td>
<td></td>
</tr>
<tr>
<td>x % y</td>
<td>remainder of x ÷ y</td>
<td>(2)</td>
<td></td>
</tr>
<tr>
<td>¬x</td>
<td>x negated</td>
<td></td>
<td></td>
</tr>
<tr>
<td>abs(x)</td>
<td>absolute value or magnitude of x</td>
<td></td>
<td>abs()</td>
</tr>
<tr>
<td>ceil(x)</td>
<td>x converted to integer</td>
<td>(3)</td>
<td>int()</td>
</tr>
<tr>
<td>floor(x)</td>
<td>x converted to floating point</td>
<td>(4)</td>
<td>float()</td>
</tr>
<tr>
<td>complex(x; y)</td>
<td>a complex number with real part x, imaginary part y</td>
<td>(5)</td>
<td>complex()</td>
</tr>
<tr>
<td>copysign(x; y)</td>
<td>conjugate of the complex number c</td>
<td>(6)</td>
<td></td>
</tr>
<tr>
<td>divmod(x; y)</td>
<td>the pair (x // y, x % y)</td>
<td>(2)</td>
<td>divmod()</td>
</tr>
<tr>
<td>pow(x; y)</td>
<td>x to the power y</td>
<td>(1)</td>
<td>pow()</td>
</tr>
<tr>
<td>x ** y</td>
<td>x to the power y</td>
<td>(3)</td>
<td></td>
</tr>
</tbody>
</table>

All string, bool, and class types also include the following operations:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
<th>Notes</th>
<th>Full documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>math.trunc(x)</td>
<td>a truncated to intpair</td>
<td></td>
<td></td>
</tr>
<tr>
<td>round(x, d)</td>
<td>x rounded to d-digits, rounding half to even. If d is omitted, it defaults to 0.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>math.ceil(x)</td>
<td>the greatest integer &lt;= x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>math.floor(x)</td>
<td>the least integer &gt;= x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For additional numeric operations see the math and cmath modules.
how can we combine elements?

expressions are combinations of values (operands) and operators, that can be evaluated and return a result

For example,

```python
>>> 99 + 100
199
```

```python
>>> 99.9 + 100
199.9
```

```python
>>> len(10)
```

Python is a strongly typed language

Some operators are overloaded

a value is an expression:

```python
>>> 99
99
```

```python
>>> a = 100
```

```python
>>> a
```

order of precedence for operators:

```python
ctemp = ftemp - 32 * 5/9
```
Python overview – a closer look
✓ what are the primitive elements?
  – objects with type, value, id (memory location)
✓ how can we combine elements?
  – using expressions that return values

• how can we create our own elements?
  – naming values (variables/assignment)
  – user-defined functions
  – user-defined data types

For example,
```python
>>> x = 10
```

Python processes an assignment statement by:
1. evaluating the expression on the rhs
2. associating name on lhs with resulting value

Why variables?

-- documentation
-- re-use

3.14 * 2**2

pi = 3.14
r = 2
circumference = 2 * pi * r
area = pi * r**2
Variable assignment

<variable> = <expression>

>>> b = 20

No value is returned. Assignment statements are not expressions; nothing is evaluated.

The identifier is added to the Python namespace.

>>> b
# Recall: identifiers (variable names)

??
# are expressions

assigning a value to a variable does not change any other variable
CIS 210

Assignment statements

```python
>>> a = 10
>>> c = a
>>> a = a * 10
>>> c
>>> a
?? ??
```

---

CIS 210

```python
>>> x = 10
>>> x
10 a variable is a named reference
>>> type(x)
<class 'int'> (pointer) to a data object
>>> id(x)
4297261440
```

---

CIS 210

```python
>>> x = x + 10
>>> x
20 when the variable is given a new value
>>> type(x)
<class 'int'> the reference (pointer) is updated
>>> id(x)
4297261760
```

---

CIS 210

- Where do I start? A step-by-step guide to computational problem-solving: how to write a good, working Python program
  - Python overview – a closer look
    - primitive elements
    - combining primitive elements
    -- expressions -- strong typing
    -- overloaded operators -- operator precedence
    -- extending the language
      - variable assignment
      -- Python variables are references
      -- dynamic typing
      -- accumulator pattern
      -- Python repeat (for)
  - Review/Project 1 Solution
  - How to start Project 2
A variable is a named reference (pointer) to a data object.

```python
>>> x = 10.5
>>> x
10.5
>>> type(x)
<class 'float'>
>>> id(x)
4300280912
```

```python
>>> x = x + 10
>>> x
20.5
>>> id(x)
4300282592
>>> id(y)
4300280912
```

garbage collection

- Python interpreter
- check whether any variables are pointing at objects for which it has allocated memory
- if none are, the object is deleted and the memory is made available again
Python is a dynamically typed language

```
a = 10
b = a
a = 'hello, world'
```

```
>>> type(a)  >>> type(b)
??          ??
```

static typing

```
var a : int
a = 4
a = 'hello' X
```

dynamic typing

```
greenCt = 1
for ctr in range(4):
    greenCt = greenCt + 1
print(greenCt)
```

```
greenCt = 1
for ctr in range(4):
    greenCt = greenCt + 1  #greenCt += 1
print(greenCt)
```

accumulator pattern
CIS 210

accumulator pattern

• initialize accumulator variable
• repeatedly adjust the accumulator variable
• until done

CIS 210

accumulator pattern

recall:
\[ x = 1 \]
\[ x = x + 1 \]
repeat 2 times:
\[ x = x + 1 \]

CIS 210

Repeat operation in Python

for <var> in range (<int>):
    <do something>

# x += 1 better yet
Repeat operation in Python

```python
for <var> in range <int>:
    <do something>
```

```python
for item in range(2):
    print(item)
```

Python Shell:
```
>>> range(2) # returns a range object
range(0, 2)
>>> list(range(2)) # that looks like this
[0, 1]
```

Repeat operation in Python

```python
for item in range(2):
    print(item)
```

```
x = 99
n = 5
for _ in range(n):
    x = x + 1 # x += 1
    print(x)
```

?? – what is the result when this code is executed?

```
for i in range(1, n + 1, 2):
    x = x + i # x += i
    print(x)
```

?? – what is the result when this code is executed?
CIS 210
Projects – Getting Started

Review and understand project specification –
**mysqrt, sqrt_compare, main**

main calls sqrt_compare calls mysqrt

start with the lowest level function

CIS 210
Projects – Getting Started

**mysqrt**

**mysqrt** will have two parameters, n, the number to find the square root for, and k, the number of times the iterative square root approximation process should run. The function should return the approximate square root value for n.

CIS 210

Develop thorough understanding of the algorithm:

\[ X_{k+1} = \frac{1}{2} \times \left( X_k + \frac{n}{X_k} \right), \text{where } X_0 = 1 \]
Develop thorough understanding of the algorithm:

\[ x_{k+1} = \frac{1}{2} \times \left( x_k + \frac{n}{x_k} \right), \text{where } x_0 = 1 \]

for \( n = 4 \) and \( k = 3 \)

\[
\begin{align*}
    x_0 &= 1 \\
    x_1 &= \frac{1}{2} \times \left( x_0 + \frac{n}{x_0} \right) = \frac{1}{2} \times (1 + 4/1) = 2.5 \\
    x_2 &= \frac{1}{2} \times \left( x_1 + \frac{n}{x_1} \right) = \frac{1}{2} \times (2.5 + 4/2.5) = 2.05 \\
    x_3 &= \frac{1}{2} \times \left( x_2 + \frac{n}{x_2} \right) = \frac{1}{2} \times (2.05 + 4/2.05) = 2.0006
\end{align*}
\]

Iterative method: a first approximation is produced, then a method which improves the accuracy of the solution accuracy of the solution is used for a certain number of iterations or until two successive approximations agree to the accuracy required.

\[ x_{k+1} = \frac{1}{2} \times \left( x_k + \frac{n}{x_k} \right), \text{where } x_0 = 1 \]
for n = 4 and k = 3

\[ x_0 = 1 \]
\[ x_1 = .5 \times (x_0 + n/x_0) \]
\[ x_2 = .5 \times (x_1 + n/x_1) \]
\[ x_3 = .5 \times (x_2 + n/x_2) \]

\[ x_0 = 1 \]
\[ x_1 = .5 \times (1 + 4/1) = 2.5 \]
\[ x_2 = .5 \times (2.5 + 4/2.5) = 2.05 \]
\[ x_3 = .5 \times (2.05 + 4/2.05) = 2.0006 \]

Note:
\[ x_{k+1} = x_k + 1 \]
\[ x_{\text{new}} = x_{\text{old}} + 1 \]

?? in Python

Note:
\[ x_{k+1} = x_k + 1 \]  
\[ x_{\text{new}} = x_{\text{old}} + 1 \]  
\[ x = x + 1 \]  

accumulator pattern