Recall: 3 important questions you should ask about any programming language:

• what are the primitive elements?
  – objects – type, value, id (memory location)
• how can we combine elements?
  – using expressions that return values
• how can we create our own elements?

**Expressions** are combinations of values (operands) and operators (functions), that can be evaluated and return a result.

For example,

```python
>>> 99 + 100
199
>>> 99.9 + 100
??
```

```python
>>> round(150.1)
150
>>> ctemp_to_f(temp(100))
212.0
```
3 important questions you should ask about any programming language:

- **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
```

Why variables?
- documentation
- re-use

```python
>>> 3.14 * 2**2
>>> pi = 3.14
>>> radius = 2
>>> area = pi * r**2
>>> area
```
how can we create our own elements?
✓ naming values (variables/assignment)
  – user-defined functions

For example
```python
def twice():  # header incl. parms
    """
    # docstring
    """
    result = 3 * 2  # code
    return result  # value to return
```

Defining a function is like defining a variable name –

the function name refers to the function value (the body of the function).

Functions are an executable data type.
```python
def twice():
    """
    """
    result = 3 * 2
    return result
```

They must be called:
```python
>>> twice()
6
```
```
def twice(x):
    """
    result = x * 2
    return result
    """

>>> twice(3)
3 is an argument
>>> twice(5)
5 is an argument
```
def twice(x):
    ""
    result = x * 2
    return result

>>> twice(100)  x = ?  return value is ?
200

>>> twice(1)  x = ?  return value is ?
2

def twice(x):
    ""
    result = x * 2
    return result

>>> twice(100)  x = 100  return value is 200

>>> twice(1)  x = 1  return value is 2

def twice(x):
    ""
    result = 2 * x
    return None

>>> twice(5)  result = 2 * x

>>> twice(5)  return None

def twice(x):
    ""
    result = 2 * x
    print(result)
    return result

>>> twice(5)  result = 2 * x

>>> twice(5)  print(result)

>>> twice(5)  return result

??

??

??

??
Functions ALWAYS return a value
(sometimes the value is None)

---

def twice(x):
    ''''
    result = 2 * x
    return result

def twice(x):
    ''''
    result = 2 * x
    return 99

def twice(x):
    ''''
    result = 2 * x
    return None

***

def doubleDouble(x):
    ''''
    result = twice(x) + twice(x)
    return result

>>> doubleDouble(4)
>>> ??

---

def twice(x):
    ''''
    x = 3
    result = x * 2
    return result

>>> twice(4)
??

---
Keeping track – Python tools so far

1. numeric data types (int and float)/operations
2. expressions
3. assignment statements
4. user-defined functions
5. → Python repeat

For example: approximating the square root of a number (positive integer n)

the algorithm is given:

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

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

... [k times]

$$x = 1$$
$$x = .5 \times (x + n/x)$$

... [k times]

repeat k times:

$$x = .5 \times (x + n/x)$$
Repeat operation in Python

for item in range(rnum):
    <do something>

Python Shell:

```python
>>> range(4)  # returns a range object
range(0, 4)
```

```python
>>> list(range(4))  # which looks like this
[0, 1, 2, 3]
```
\[
x = 1 \\
\text{repeat } k \text{ times:} \\
x = 0.5 \times (x + n/x) \\
x = 1 \\
\text{for } \text{ctr in range}(k): \quad \# \text{implements repeat} \\
x = 0.5 \times (x + n/x) \quad \# k \text{ times}
\]

**Boolean Expressions/Conditional Statements**

if <boolean expression>:
  <block of code>

<next Python statement>

**Selection/Conditional Statement: Flow of Control**

- False
- Condition
- True
- Statements

**Keeping track – Python tools so far**

1. numeric data types (int and float)/operations
2. expressions
3. assignment statements
4. user-defined functions
5. Python repeat (for loop)
6. → conditional statements
7. → Boolean data type
Figure 2.11

Flow of control

if <boolean expression>:
    <block of code>
elsif <boolean expression>:
    <block of code>
elsif <boolean expression>:
    <block of code>
else:
    <block of code>
<next Python statement>

Boolean expressions

logical/relational operators

return a Boolean value

True
False
### Boolean Expressions

<table>
<thead>
<tr>
<th>Condition</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>a &lt; b</td>
<td>not a &lt; b</td>
</tr>
<tr>
<td>a &lt;= b</td>
<td>a &lt;= b and c &gt;= d</td>
</tr>
<tr>
<td>a &gt; b</td>
<td>a &lt;= b or c &gt;= d</td>
</tr>
<tr>
<td>a == b</td>
<td></td>
</tr>
<tr>
<td>a != b</td>
<td></td>
</tr>
</tbody>
</table>

### Order of Precedence:
- Relational operators: >
- Logical operators: and, or
- PARENS (best)

### Keeping track – Python tools

1. Numeric data types (int and float) / operations
2. Expressions
3. Assignment statements
4. User-defined functions
5. Python repeat (for)
6. Conditional statements
7. Boolean data type
8. Python standard library / import (lab) (including turtle graphics)

### Monte Carlo Algorithms

- Statistical simulation methods, where statistical simulation is defined in quite general terms to be any method that utilizes sequences of random numbers to perform the simulation

- Any method which solves a problem by generating suitable random numbers and observing that fraction of the numbers obeying some property or properties
Monte Carlo Simulation to Approximate Pi

- Simulation of a game of darts
- Randomly place darts on the board
- Value of pi can be computed by keeping track of the number of darts that land on the board

Monte Carlo Simulation (Problem 2-2)

- the fraction of darts that lands in the circle is \( \frac{\pi/4}{1} = \pi/4 \)
- the fraction of darts that lands in the circle is \( \frac{\text{inCircleCt}}{\text{numDarts}} \)
- \( \text{inCircleCt} / \text{numDarts} = \pi / 4 \rightarrow \pi = 4 \times \left( \frac{\text{inCircleCt}}{\text{numDarts}} \right) \)
- to determine whether a dart has landed in the circle – use formula for finding the distance between the point and the origin:
  \[ d = \sqrt{x^2 + y^2} \]
- how do we throw darts at the board??
Monte Carlo Simulation (Problem 2-2)

- the fraction of darts that lands in the circle is \( \frac{\pi}{4} \)
- the fraction of darts that lands in the circle is \( \frac{\text{inCircleCnt}}{\text{numDarts}} \)
- \( \frac{\text{inCircleCnt}}{\text{numDarts}} = \frac{\pi}{4} \rightarrow \pi = 4 \times \left( \frac{\text{inCircleCnt}}{\text{numDarts}} \right) \)
- to determine whether a dart has landed in the circle – use formula for finding the distance between the point and the origin:
  \[ d = \sqrt{x^2 + y^2} \]
- generate \( x \) and \( y \) – use formula for finding the distance between the point and the origin:

```python
>>> help(random.random)
Help on built-in function random:
 random() -> x in the interval [0, 1).
```

Monte Carlo Simulation (Problem 2-2)

(0) type in the `montePi` function from the text

(1) add docstring per CIS 210 style guidelines

(2) modify `montePi` so that it calls a new `isInCircle` function (exercises 2.38 and 2.39); add the docstring

(3) revise `showMontePi` starter code (from text or class website or `montePi`) so that it calls `isInCircle`; add the docstring

(4) modify `showMontePi` to report on the error in the approximation of \( \pi \) produced by the function (as for the approximate square root function)

CIS 210 / Welcome

- Introduction to Python
  - Python objects
  - numeric data types and operations
  - expressions
  - REPL (lab)
  - variables and assignment
  - user-defined functions
  - calling/executing a function
  - function return values and side effects
  - Python standard library / import (lab)
  - Python turtle graphics (lab)
  - Python repeat (for)
  - Boolean data type
  - Python conditionals

- Monte Carlo algorithm for approximating \( \pi \)
- Designing Computational Solutions to Problems
- CIS210 Programming Style Guidelines (Python docstring)