Announcements

☐ Hog revisions due Monday

☐ HW10 due Wednesday

☐ Make sure to fill out survey on Piazza
  ☐ We need to schedule alternate final exam times for those who have a conflict, so if you do, let us know on the survey when you are available
The Begin Special Form

Begin expressions allow sequencing
The Begin Special Form

Begin expressions allow sequencing

\( (\text{begin} \ <\text{exp}_1> \ <\text{exp}_2> \ \ldots \ <\text{exp}_n>) \)
Begin expressions allow sequencing

\[
\text{(begin } \langle \text{exp}_1 \rangle \ \langle \text{exp}_2 \rangle \ \ldots \ \langle \text{exp}_n \rangle)\]

\[
\text{(define } (\text{repeat } k \ \text{fn})\)\]
Begin expressions allow sequencing

\[(\text{begin} \ <\text{exp}_1>\ <\text{exp}_2>\ \ldots\ <\text{exp}_n>)\]

\[(\text{define} \ (\text{repeat} \ k \ \text{fn})\]
\[(\text{if} \ (> \ k \ 0)\]
The Begin Special Form

Begin expressions allow sequencing

\[
\text{(begin } \langle \text{exp}_1 \rangle \ \langle \text{exp}_2 \rangle \ \ldots \ \langle \text{exp}_n \rangle \text{)}
\]

\[
\text{(define } (\text{repeat k fn}) \\
\quad (\text{if } (> k 0) \\
\quad \quad (\text{begin } (\text{fn}) \ (\text{repeat } (- k 1) \text{ fn}))
\]

The Begin Special Form

Begin expressions allow sequencing

\[
\text{(begin } \exp_1 \exp_2 \ldots \exp_n)\]

\[
\text{(define } \text{(repeat } k \text{ fn)} \text{)}
\[
\quad \text{(if } (> k 0) \text{)}
\[
\quad \quad \text{(begin } \text{(fn)} \text{ (repeat } (- k 1) \text{ fn))}
\[
\quad \quad \text{'done})\]
The Begin Special Form

Begin expressions allow sequencing

\[(\text{begin} \; \langle \text{exp}_1 \rangle \; \langle \text{exp}_2 \rangle \; \ldots \; \langle \text{exp}_n \rangle)\]

\[(\text{define} \; (\text{repeat} \; k \; \text{fn})\]
\[(\text{if} \; (> \; k \; 0)\]
\[\; (\text{begin} \; (\text{fn}) \; (\text{repeat} \; (- \; k \; 1) \; \text{fn}))\]
\[\; \langle \text{done} \rangle)\]

\[(\text{define} \; (\text{tri} \; \text{fn})\]
The Begin Special Form

Begin expressions allow sequencing

\[(\text{begin } \exp_1 \exp_2 \ldots \exp_n)\]

(\text{define (repeat k fn)}
 (\text{if (> k 0)}
  \ (\text{begin (fn) (repeat (- k 1) fn)})
  'done))

(\text{define (tri fn)}
 (\text{repeat 3 (lambda () (fn) (lt 120))}))
The Begin Special Form

Begin expressions allow sequencing

\[(\text{begin } \exp_1\ exp_2\ \ldots\ \exp_n)\]

\[(\text{define} (\text{repeat} k \ fn))\]
\[\quad (\text{if} \ (> k 0)\]
\[\quad \quad (\text{begin} \ (fn) \ (\text{repeat} (- k 1) \ fn))\]
\[\quad \quad '\text{done})\]

\[(\text{define} (\text{tri} fn)\]
\[\quad (\text{repeat} 3 \ (\text{lambda} ()\ (fn) \ (\text{lt} 120))))\]

\[(\text{define} (\text{sier} d k)\]
Begin Expressions allow sequencing

\[
\text{(begin } \exp_1\text{ } \exp_2\text{ } \ldots \exp_n)\text{)}
\]

\[
(\text{define } (\text{repeat } k \text{ fn})
\text{ }
(\text{if } (> \text{ k } 0)
\text{ }
(\text{begin } (\text{fn}) (\text{repeat } (- \text{ k } 1) \text{ fn}))
\text{ 'done}))
\]

\[
(\text{define } (\text{tri } \text{ fn})
\text{ }
(\text{repeat } 3 (\lambda () (\text{fn}) (\text{lt } 120))))
\]

\[
(\text{define } (\text{sier } d \text{ k})
\text{ }
(\text{tri} (\lambda () (\text{if } (= \text{ k } 1) (\text{fd } d) (\text{leg } d \text{ k}))))
\)
The Begin Special Form

Begin expressions allow sequencing

\[
\text{(begin } \exp_1 \exp_2 \ldots \exp_n)\]

\[
\begin{align*}
(\text{define } & (\text{repeat } k \ fn) \\
& (\text{if } (> k 0) \\
& \quad \text{(begin } (fn) \text{ (repeat } (- k 1) \ fn) \text{))} \\
& \quad \text{'done})
\end{align*}
\]

\[
\begin{align*}
(\text{define } & (\text{tri } fn) \\
& (\text{repeat } 3 \ (\text{lambda } () \ (fn) \ (\text{lt } 120)))
\end{align*}
\]

\[
\begin{align*}
(\text{define } & (\text{sier } d \ k) \\
& (\text{tri } (\text{lambda } () \ (\text{if } (= k 1) \ (fd \ d) \ (\text{leg } d \ k))))
\end{align*}
\]

\[
(\text{define } (\text{leg } d \ k)
\]
The Begin Special Form

Begin expressions allow sequencing

\[(\text{begin} \ <\text{exp}_1> \ <\text{exp}_2> \ \ldots \ <\text{exp}_n>\)]

\[(\text{define} \ (\text{repeat} \ k \ \text{fn})\)
   \quad (\text{if} \ (> \ k \ 0)\)
   \quad \quad (\text{begin} \ (\text{fn}) \ (\text{repeat} \ (- \ k \ 1) \ \text{fn})\)
   \quad \quad '\text{done})\]

\[(\text{define} \ (\text{tri} \ \text{fn})\)
   \quad (\text{repeat} \ 3 \ (\text{lambda} () \ (\text{fn}) \ (\text{lt} \ 120)))\]

\[(\text{define} \ (\text{sier} \ d \ k)\)
   \quad (\text{tri} \ (\text{lambda} () \ (\text{if} \ (= \ k \ 1) \ (\text{fd} \ d) \ (\text{leg} \ d \ k)))\))\]

\[(\text{define} \ (\text{leg} \ d \ k)\)
   \quad (\text{sier} \ (/ \ d \ 2) \ (- \ k \ 1)) \ (\text{penup}) \ (\text{fd} \ d) \ (\text{pendown}))\]
Handling Errors (Back to Python)
Handling Errors (Back to Python)

Sometimes, computers don't do exactly what we expect
Handling Errors (Back to Python)

Sometimes, computers don't do exactly what we expect

• A function receives unexpected argument types
Handling Errors (Back to Python)

Sometimes, computers don't do exactly what we expect

- A function receives unexpected argument types
- Some resource (such as a file) is not available
Handling Errors (Back to Python)

Sometimes, computers don't do exactly what we expect

• A function receives unexpected argument types
• Some resource (such as a file) is not available
• A network connection is lost
Handling Errors (Back to Python)

Sometimes, computers don't do exactly what we expect

• A function receives unexpected argument types
• Some resource (such as a file) is not available
• A network connection is lost

September 9 1947: Moth found in a Mark II Computer
Exceptions
Exceptions

A built-in mechanism in a programming language to declare and respond to exceptional conditions
Exceptions

A built-in mechanism in a programming language to declare and respond to exceptional conditions

Python *raises* an exception whenever an error occurs
Exceptions

A built-in mechanism in a programming language to declare and respond to exceptional conditions

Python raises an exception whenever an error occurs

Exceptions can be handled by the program, preventing a crash
Exceptions

A built-in mechanism in a programming language to declare and respond to exceptional conditions

Python raises an exception whenever an error occurs

Exceptions can be handled by the program, preventing a crash

Unhandled exceptions will cause Python to halt execution
Exceptions

A built-in mechanism in a programming language to declare and respond to exceptional conditions

Python *raises* an exception whenever an error occurs

Exceptions can be *handled* by the program, preventing a crash

Unhandled exceptions will cause Python to halt execution

**Mastering exceptions:**
Exceptions

A built-in mechanism in a programming language to declare and respond to exceptional conditions

Python *raises* an exception whenever an error occurs

Exceptions can be *handled* by the program, preventing a crash

Unhandled exceptions will cause Python to halt execution

**Mastering exceptions:**

Exceptions are objects! They have classes with constructors
Exceptions

A built-in mechanism in a programming language to declare and respond to exceptional conditions

Python *raises* an exception whenever an error occurs

Exceptions can be *handled* by the program, preventing a crash

Unhandled exceptions will cause Python to halt execution

**Mastering exceptions:**

Exceptions are objects! They have classes with constructors

They enable non-local continuations of control:
Exceptions

A built-in mechanism in a programming language to declare and respond to exceptional conditions

Python *raises* an exception whenever an error occurs

Exceptions can be *handled* by the program, preventing a crash

Unhandled exceptions will cause Python to halt execution

**Mastering exceptions:**

Exceptions are objects! They have classes with constructors

They enable non-local continuations of control:

If \( f \) calls \( g \) and \( g \) calls \( h \), exceptions can shift control from \( h \) to \( f \) without waiting for \( g \) to return
Exceptions

A built-in mechanism in a programming language to declare and respond to exceptional conditions

Python *raises* an exception whenever an error occurs

Exceptions can be *handled* by the program, preventing a crash

Unhandled exceptions will cause Python to halt execution

**Mastering exceptions:**

Exceptions are objects! They have classes with constructors

They enable non-local continuations of control:

If \( f \) calls \( g \) and \( g \) calls \( h \), exceptions can shift control from \( h \) to \( f \) without waiting for \( g \) to return

However, exception handling tends to be slow
Assert statements raise an exception of type `AssertionError`
Assert Statements

Assert statements raise an exception of type `AssertionError`

```python
assert <expression>, <string>
```
Assert Statements

Assert statements raise an exception of type `AssertionError`

```
assert <expression>, <string>
```

Assertions are designed to be used liberally and then disabled in production systems
Assert Statements

Assert statements raise an exception of type `AssertionError`

```python
assert <expression>, <string>
```

Assertions are designed to be used liberally and then disabled in production systems

```
python3 -O
```
**Assert Statements**

Assert statements raise an exception of type `AssertionError`.

```
assert <expression>, <string>
```

Assertions are designed to be used liberally and then disabled in production systems.

```
python3 -O
```

"O" stands for optimized. Among other things, it disables assertions.
Assert Statements

Assert statements raise an exception of type `AssertionError`

```python
assert <expression>, <string>
```

Assertions are designed to be used liberally and then disabled in production systems

```
python3 -O
```

"O" stands for optimized. Among other things, it disables assertions

Whether assertions are enabled is governed by the built-in bool `__debug__`
Raise Statements
Raise Statements

Exceptions are raised with a *raise statement*
Raise Statements

Exceptions are raised with a *raise statement*

```
raise <expression>
```
Exceptions are raised with a *raise statement*

\[ \text{raise}\ <\text{expression}> \]

<expression> must evaluate to an exception instance or class.
Raise Statements

Exceptions are raised with a *raise statement*

```
raise <expression>
```

`<expression>` must evaluate to an exception instance or class.

Exceptions are constructed like any other object; they are just instances of classes that inherit from `BaseException`
Raise Statements

Exceptions are raised with a *raise statement*

```
raise <expression>
```

*<expression>* must evaluate to an exception instance or class.

Exceptions are constructed like any other object; they are just instances of classes that inherit from `BaseException`

**TypeError** -- A function was passed the wrong number/type of argument
Raise Statements

Exceptions are raised with a *raise statement*

```
raise <expression>
```

*<expression>* must evaluate to an exception instance or class.

Exceptions are constructed like any other object; they are just instances of classes that inherit from `BaseException`

- **TypeError** -- A function was passed the wrong number/type of argument
- **NameError** -- A name wasn't found
Exceptions are raised with a *raise statement* :

```
raise <expression>
```

*<expression>* must evaluate to an exception instance or class.

Exceptions are constructed like any other object; they are just instances of classes that inherit from `BaseException`:

- `TypeError` -- A function was passed the wrong number/type of argument
- `NameError` -- A name wasn't found
- `KeyError` -- A key wasn't found in a dictionary
Raise Statements

Exceptions are raised with a *raise statement*

```python
raise <expression>
```

*<expression>* must evaluate to an exception instance or class.

Exceptions are constructed like any other object; they are just instances of classes that inherit from `BaseException`

- **TypeError** -- A function was passed the wrong number/type of argument
- **NameError** -- A name wasn't found
- **KeyError** -- A key wasn't found in a dictionary
- **RuntimeError** -- Catch-all for troubles during interpretation
Try Statements
Try Statements

*Try statements* handle exceptions
Try statements handle exceptions

```python
try:
    <try suite>
except <exception class> as <name>:
    <except suite>
...```

Try statements
Try Statements

*Try statements* handle exceptions

```
try:
    <try suite>
except <exception class> as <name>:
    <except suite>
...
```

Execution rule:
Try Statements

*Try statements* handle exceptions

```
try:
    <try suite>
except <exception class> as <name>:
    <except suite>
...
```

Execution rule:

- The `<try suite>` is executed first;
Try Statements

*Try statements* handle exceptions

```python
try:
    <try suite>
except <exception class> as <name>:
    <except suite>
...
```

Execution rule:

- The *<try suite>* is executed first;
- If, during the course of executing the *<try suite>*
  an exception is raised that is not handled otherwise, and
Try Statements

Try statements handle exceptions

```python
try:
    <try suite>
except <exception class> as <name>:
    <except suite>
...```

Execution rule:

• The `<try suite>` is executed first;
• If, during the course of executing the `<try suite>`, an exception is raised that is not handled otherwise, and
• If the class of the exception inherits from `<exception class>`, then
Try Statements

*Try statements* handle exceptions

```python
try:
    <try suite>
except <exception class> as <name>:
    <except suite>
...
```

Execution rule:

- The `<try suite>` is executed first;
- If, during the course of executing the `<try suite>`, an exception is raised that is not handled otherwise, and
- If the class of the exception inherits from `<exception class>`, then
- The `<except suite>` is executed, with `<name>` bound to the exception
Handling Exceptions
Handling Exceptions

Exception handling can prevent a program from terminating.
Exception handling can prevent a program from terminating

>>> try:
Handling Exceptions

Exception handling can prevent a program from terminating

```python
>>> try:
    x = 1/0
```
Handling Exceptions

Exception handling can prevent a program from terminating

```python
>>> try:
    x = 1/0
except ZeroDivisionError as e:
```
Handling Exceptions

Exception handling can prevent a program from terminating

```python
>>> try:
    x = 1/0
except ZeroDivisionError as e:
    print('handling a', type(e))
```
Handling Exceptions

Exception handling can prevent a program from terminating

```python
>>> try:
    x = 1/0
except ZeroDivisionError as e:
    print('handling a', type(e))
    x = 0
```
Handling Exceptions

Exception handling can prevent a program from terminating

```python
>>> try:
    x = 1/0
except ZeroDivisionError as e:
    print('handling a', type(e))
    x = 0

handling a <class 'ZeroDivisionError'>
```
Handling Exceptions

Exception handling can prevent a program from terminating

```python
>>> try:
    x = 1/0
except ZeroDivisionError as e:
    print('handling a', type(e))
    x = 0

handling a <class 'ZeroDivisionError'>
>>> x
```

Handling Exceptions

Exception handling can prevent a program from terminating

```python
>>> try:
    x = 1/0
    except ZeroDivisionError as e:
        print('handling a', type(e))
    x = 0

handling a <class 'ZeroDivisionError'>

>>> x
0
```
Exception handling can prevent a program from terminating

```python
>>> try:
    x = 1/0
    except ZeroDivisionError as e:
        print('handling a', type(e))
    x = 0

handling a <class 'ZeroDivisionError'>
```

Multiple try statements: Control jumps to the except suite of the most recent try statement that handles that type of exception.
How will the Python interpreter respond?
WWPD: What Would Python Do?

How will the Python interpreter respond?
How will the Python interpreter respond?

```python
def invert(x):
    result = 1/x  # Raises a ZeroDivisionError if x is 0
    print('Never printed if x is 0')
    return result

def invert_safe(x):
    try:
        return invert(x)
    except ZeroDivisionError as e:
        return str(e)
```
How will the Python interpreter respond?

```python
def invert(x):
    result = 1/x  # Raises a ZeroDivisionError if x is 0
    print('Never printed if x is 0')
    return result

def invert_safe(x):
    try:
        return invert(x)
    except ZeroDivisionError as e:
        return str(e)

>>> invert_safe(1/0)
```

WWPD? What Would Python Do?
def invert(x):
    result = 1/x  # Raises a ZeroDivisionError if x is 0
    print('Never printed if x is 0')
    return result

def invert_safe(x):
    try:
        return invert(x)
    except ZeroDivisionError as e:
        return str(e)

>>> invert_safe(1/0)
>>> try:
How will the Python interpreter respond?

```python
def invert(x):
    result = 1/x  # Raises a ZeroDivisionError if x is 0
    print('Never printed if x is 0')
    return result

def invert_safe(x):
    try:
        return invert(x)
    except ZeroDivisionError as e:
        return str(e)

>>> invert_safe(1/0)
>>> try:
    invert_safe(0)
```

WWPD: What Would Python Do?
How will the Python interpreter respond?

```python
def invert(x):
    result = 1/x  # Raises a ZeroDivisionError if x is 0
    print('Never printed if x is 0')
    return result

def invert_safe(x):
    try:
        return invert(x)
    except ZeroDivisionError as e:
        return str(e)

>>> invert_safe(1/0)
```

```
>>> try:
    invert_safe(0)
except BaseException:
```
def invert(x):
    result = 1/x  # Raises a ZeroDivisionError if x is 0
    print('Never printed if x is 0')
    return result

def invert_safe(x):
    try:
        return invert(x)
    except ZeroDivisionError as e:
        return str(e)

>>> invert_safe(1/0)
>>> try:
    invert_safe(0)
except BaseException:
    print('Handled!')
How will the Python interpreter respond?

```python
def invert(x):
    result = 1/x  # Raises a ZeroDivisionError if x is 0
    print('Never printed if x is 0')
    return result

def invert_safe(x):
    try:
        return invert(x)
    except ZeroDivisionError as e:
        return str(e)

>>> invert_safe(1/0)
>>> try:
    invert_safe(0)
except BaseException:
    print('Handled!')

>>> inverrrrrt_safe(1/0)
```