A closer look:

- Python objects
- Python assignment

-- mental model of computational processes
-- for reading/writing more advanced code
-- for analyzing (preventing!) errors
-- programming language(s)

Assignment statements

Recall:

dynamic typing (Python) v. static typing

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

>>> 99 + 10
??

Recall: we can combine objects in expressions, which are evaluated and return a value

For example,

```python
>>> 99 + 10
??
>>> len('hello')
??
>>> str.center('****', 10)
??
```

Python is a strongly typed language
Recall: we can combine objects in expressions, which are evaluated and return a value.

For example,

```
>>> len(10)
??
>>> str.center(math.pi, 10)
??
```

Python is a strongly typed language.

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

For example,

```
>>> 99.9 + 100
??
```

Python is an extremely strongly typed language.

---

a) >>> x = 10
   >>> x = 'hi'
   >>> x = -.99 < 0
   >>> approx_rt = .5 * (x + n/x)
   for _ in range(20):
       print(x)

b) n = 4
   >>> x = 1
   >>> x = 'hi'

c) >>> 'testing' + 123
   TypeError: must be str, not int

1) dynamic typing
2) operator overloading
3) strong typing
4) dynamic typing (error)

---

Python overview – a closer look

Recall:
- Python’s primitive elements

There is only one kind of primitive element in Python

OBJECTS

- what are Python’s primitive elements?

  OBJECT
  type
  value(s)
  id (memory location)
Type – range of values and operations (attributes)
Value(s)
Id – memory location

Examples of Python objects:
>>> 99
99
>>> type(99)
<class 'int'>
>>> id(99)
2276160

· Closer look: Python objects/assignment
  ✓ Dynamic typing
  ✓ Strong typing
  ✓ Python objects – value(s), type, id
    – Reference semantics
    – Garbage collection

Recall: Objects can be named

For example,
>>> x = 10

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

Variable assignment
<variable> = <expression>

>>> b = 20
>>> b
??
>>> b = 30
>>> b
??
>>> b = b + 1
>>> b
??
Variable assignment

<variable> = <expression>

```python
>>> b = 20
>>> a = b + 1
>>> a
```

1. evaluate the expression on the rhs
2. associate name on lhs with resulting value

```python
>>> b = 20
>>> b = 20
>>> b = 20
>>> a = b + 1
>>> a
>>> b = 30
>>> b = 40
>>> a
```

1. evaluate the expression on the rhs
2. associate name on lhs with resulting value

---

1. evaluate the expression on the rhs
2. associate variable name on lhs with resulting value/Python object

- allocate space in memory for the object
- search current namespace -- if name on lhs is not there:
  - assign name on lhs to address of memory location (reference)
  - add it to the current namespace
- if name on lhs is there:
  - replace old reference with new reference (address of memory location)

```python
>>> b = 20
>>> id(b)
4297645024
>>> a = b + 1
>>> id(a)
4297645056
>>> b
20
>>> a
21
>>> a == b
False
>>> id(a) == id(b)
False
>>> a is b
False
```
>>> b = 20  
>>> a = b + 1  
>>> a  
21

1. evaluate the expression on the rhs  
2. associate name on lhs with resulting value

>>> a = b + 1  
>>> a  
??

>>> a
21

>>> b = 30  
>>> a
21

>>> b = 30  
>>> a
21

Evaluate rhs. If it is a variable name:  
Let the variable on lhs refer to resulting object:  
− use address of rhs variable, no memory is allocated,  
− search current namespace − if name on lhs is not there:  
  − assign name on lhs to address of memory location  
  − add it to the current namespace  
  − if name on lhs is there:  
    − replace old reference with new reference (address)
>>> b = 20
>>> id(b)
4297645024

>>> a = b

>>> b = 20
>>> id(b)
4297645024

>>> a = b
>>> id(a)
4297645024

>>> b = 40
>>> id(b)
4297645056

>>> a = b + 1
>>> id(a)
4297645056

>>> b = 40
>>> id(b)
4297645056

>>> a

>>> id(a)
4297645056

>>> msg = 'hello'
reference semantics:

>>> msg
'hello'
the name on the lhs
is a reference (pointer)
to the memory location
of the data object

>>> greeting = msg

>>> id(msg)
4383118984

>>> id(greeting)
greeting "aliases" msg
4383118984

>>> id(msg)
4383118984

>>> id(greeting)
greeting "aliases" msg
4383118984
```python
>>> msg = 'goodbye'
>>> msg
'goodbye'
when the variable is given a new value the reference (pointer) is updated
>>> id(msg)
4383119544

>>> id(greeting)
??
```

```python
>>> msg = 'hello'
>>> msg = 'jello'
>>> class = 'CIS 210'
>>> class = 'CIS 211'
```

```python
>>> class = 'CIS 210'
>>> class = 'CIS 211'
```

```python
>>> class = 'CIS 210'
```

```python
>>> msg = 'hello'
>>> msg = 'jello'
>>> class = 'CIS 210'
>>> class = 'CIS 211'
```

```
• 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
```

```
• closer look: Python objects/assignment
  ✓ dynamic typing
  ✓ strong typing
  ✓ Python objects – value(s), type, id
  ✓ reference semantics
  ✓ garbage collection
```

```
• garbage collection
```

```
• Programming/Computer science concepts
  Computational Problem Solving: designing, implementing, checking, revising algorithms/algorithmics
  Good programming style: function documentation (type contract; description including parameters, return value, and side effects); code written in functions is reusable and may be re-used in other programs
  Coding style: indentation, comments, module documentation syntax, naming conventions (why not snake case)
  Python is a programming language and Python is an interpreter (programming)
  PythonShell is a REPL (read-evaluate-print loop)
  Python is a dynamically typed language: memory location, memory management: garbage collection, immutable data type
  Combining primitive elements: expressions – expressions evaluate to a value, short circuit evaluation of boolean expressions, overloaded operators
  Naming values: Variables/assignment: assignment statements are not expressions and do not return a value; namespacing – builtins and global (__main__) scope, dynamic typing, strong typing
  Functions are an executable data type, what happens when a Function – method – is called
  Activation record/stack frame added to call stack for local namespace; return address
  Call-by-assignment parameter passing
  Functions always return a value (sometimes None)
  Functions sometimes have side effects
  Functions as arguments
  Iterative algorithms; accumulator pattern; Monte Carlo algorithms
```

```
• garbage collection
```

```
• closer look: Python objects/assignment
  ✓ dynamic typing
  ✓ strong typing
  ✓ Python objects – value(s), type, id
  ✓ reference semantics
  ✓ garbage collection
```

```
• Programming/Computer science concepts
  Computational Problem Solving: designing, implementing, checking, revising algorithms/algorithmics
  Good programming style: function documentation (type contract; description including parameters, return value, and side effects); code written in functions is reusable and may be re-used in other programs
  Coding style: indentation, comments, module documentation syntax, naming conventions (why not snake case)
  Python is a programming language and Python is an interpreter (programming)
  PythonShell is a REPL (read-evaluate-print loop)
  Python is a dynamically typed language: memory location, memory management: garbage collection, immutable data type
  Combining primitive elements: expressions – expressions evaluate to a value, short circuit evaluation of boolean expressions, overloaded operators
  Naming values: Variables/assignment: assignment statements are not expressions and do not return a value; namespacing – builtins and global (__main__) scope, dynamic typing, strong typing
  Functions are an executable data type, what happens when a Function – method – is called
  Activation record/stack frame added to call stack for local namespace; return address
  Call-by-assignment parameter passing
  Functions always return a value (sometimes None)
  Functions sometimes have side effects
  Functions as arguments
  Iterative algorithms; accumulator pattern; Monte Carlo algorithms
```
CIS 210

Programming/Computer Science concepts, 2.

Systermic approaches to testing and debugging; automated testing

What happens when an assignment statement is executed: memory allocation; reference semantics

Parity bits

CIS 210

Python toolkit so far

numeric data types (int, float) and operations (e.g., +=, **, round, abs)
string data type and operations (e.g., +, len, count, find)
formatted strings (and string format method)
Boolean data type and relational/Boolean operations (e.g., <, and)
data type coercion functions, e.g., str, int
NoneType (None)
print, input
expressions

Python Standard Library – math, turtle, random, doctest, datetime modules; import
assignment statement
Python expressions – +, while
Python conditionals – if

variable assignment
user-defined functions; function design; docstrings
IDLE interactive development environment; help function; dir, type, id

CIS 210

A Structured Approach to Computational Problem Solving

— review the project specification thoroughly
— write examples of expected results for specified inputs — re-review spec, if needed
— develop, review, and/or review a problem-solving approach, using natural language, algorithm, pseudocode (not Python code)
— check algorithm using your examples — review algorithm, re-review spec, if needed

Starting with the lowest level function —
— write the function header
— write the function docstring — brief description
— write the function docstring — examples of use (use ones developed earlier)
— write the return statement
— using tools from the Python toolkit, start writing the body of the function
— test often, revise as needed
— test using examples in the docstring, and then project spec, and then others

CIS 210

CIS 210 Learning Outcomes

• understand, develop, implement algorithms for computational problem solving;
• use structured design and testing methods to develop and implement program;
• read, write, revise, document, test, and debug code;
• demonstrate robust mental models of data representation and code execution;
• demonstrate good understanding of a high level programming language;
• introduce and/or implement a sampling of classic computer science problem domains and algorithms.