Module 2

This is a single, concatenated file, suitable for printing or saving as a PDF for offline viewing. Please note that some animations or images may not work.

Module 2 Study Guide and Deliverables

Theme:	Basic Building Blocks for Python Programs
Readings:	• Chapter 1 (pp. 49-73), Chapter 2 (pp. 109-122), Chapter 9 (pp. 456-463), and Chapter 16 (pp. 709-722) • Module Lecture Notes
Topics:	Data Types, Hashing, Mutability, Python Ranges, Copying Objects
Assignments	Assignment 2 due on Tuesday, March 30 at 6:00 PM ET
Assessments	Quiz 2: • Available Friday, March 26 at 6:00 AM ET • Due on Tuesday, March 30 at 6:00 PM ET
Live Classrooms:	• Tuesday, March 23, 8:00 - 9:30 PM ET • Thursday, March 25, 6:00 - 7:30 PM ET • Facilitator Session: Friday, March 26, at 8:00 PM ET

Python Data Types

Python Data Types Overview

- Building blocks in a language
- Similar to noun, verb
- Python has two groups of types
    1. primitive types ("atoms")
    2. collections ("molecules")
- Additional special types:
    1. *None* type
    2. *range* type

Operators

- arithmetic:

  +, −, ∗, ∗∗, /, //, ∗∗
- assignment:

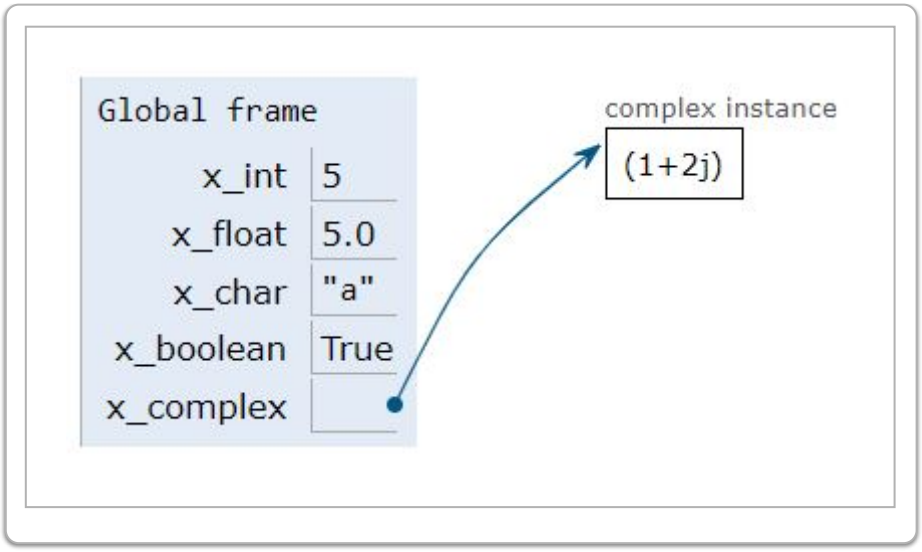
  , + =, − =, ∗ =, / =, % =, // = >
- bitwise:

  &, |,' , <<, >>
- comparison: *==, ! =, <, <=, >, >=*
- logical: *and*, *or* , *not*
- identity: is , *is not*
- membership: in , *not in*

Primitive Types

Example 1

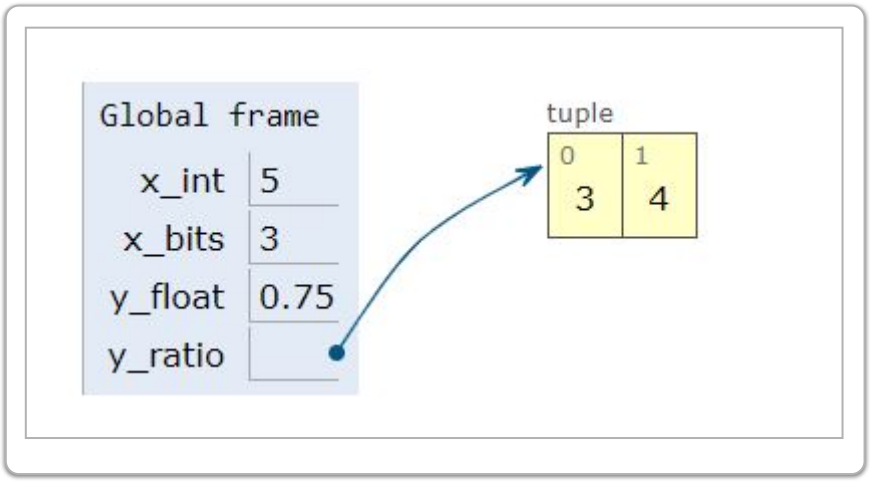
```
x_int = 5
x_float = 5.0
x_char = 'a'
x_boolean = True
x_complex = 1 + 2j
```



- 'atoms' - indivisible objects

Example 2

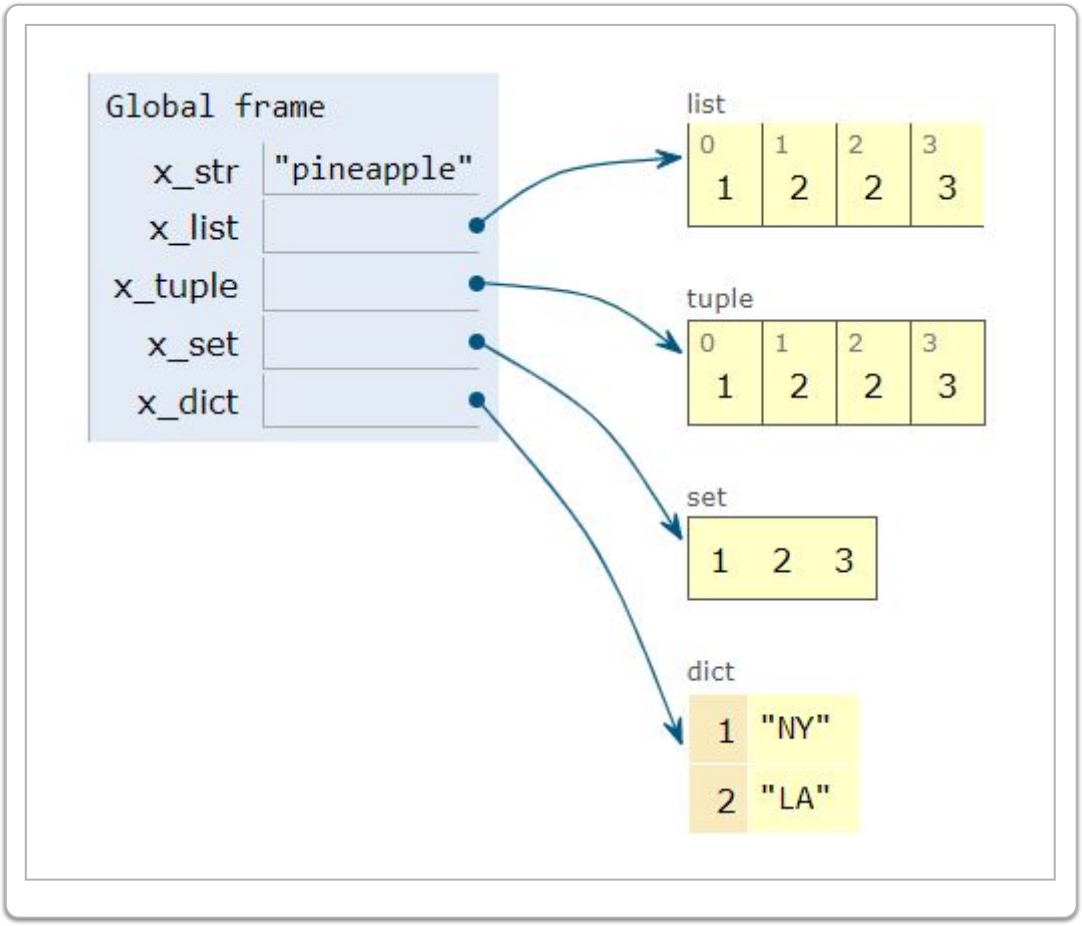
```
x_int = 5
x_bits = x_int.bit_length()

y_float = 0.75
y_ratio = y_float.as_integer_ratio()
```



- 'atoms' are not just values
- objects with methods

Collection Types

```
x_str = 'pineapple'
x_list = [1, 2, 2, 3]
x_tuple = (1, 2, 2, 3)
x_set = {1, 2, 2, 3} # note duplicates
x_dict = {1: 'NY', 2: 'LA'}
```



- 'molecules' - complex objects

Constructors for Types

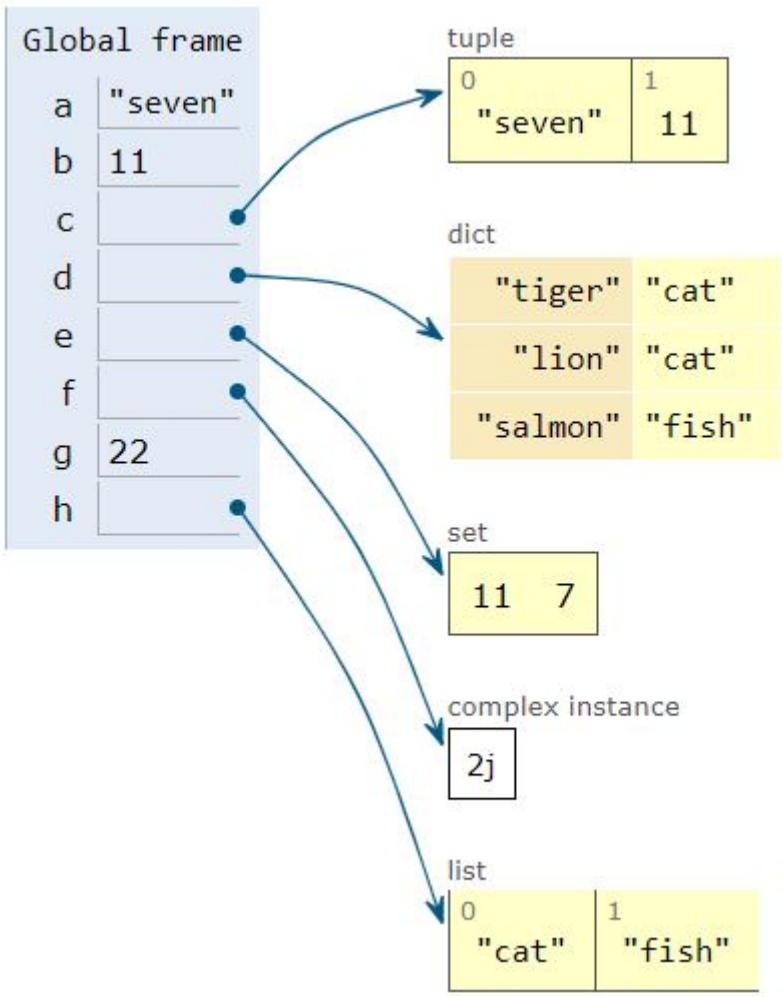
```
x_str = 'pineapple'
y_str = str('pineapple')

x_list = [1, 2, 2, 3]
y_list = list((1, 2, 2, 3))

x_tuple = (1, 2, 2, 3)
y_tuple = tuple((1, 2, 2, 3))

x_set = {1, 2, 2, 3}
y_set = set((1, 2, 2, 3))

x_dict = {1: 'NY', 2: 'LA'}
y_dict = dict({1: 'NY', 2: 'LA'})
```

- note: double brackets

Test Yourself Exercises

Test Yourself: 2.1.01

Identify the primitive and collection types.



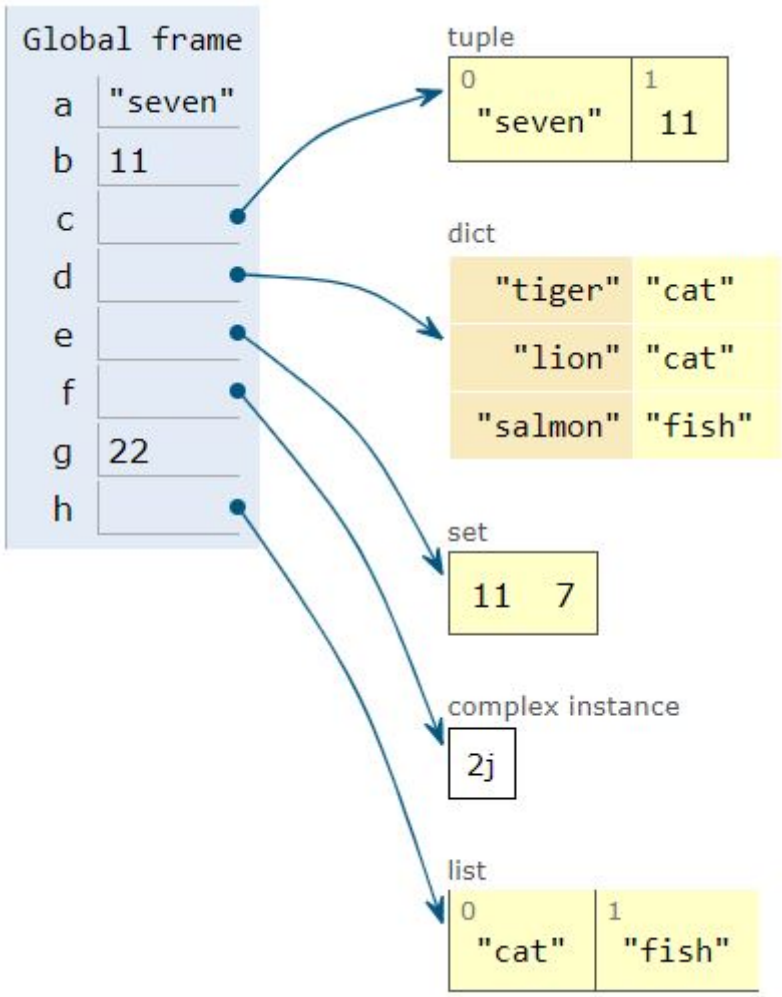
Sample Answer:

primitive types: {*b*, *f*, *g*},

collection types: {*a*, *c*, *d*, *e*, *h*}

Test Yourself: 2.1.02

Write Python code to define objects in the picture.

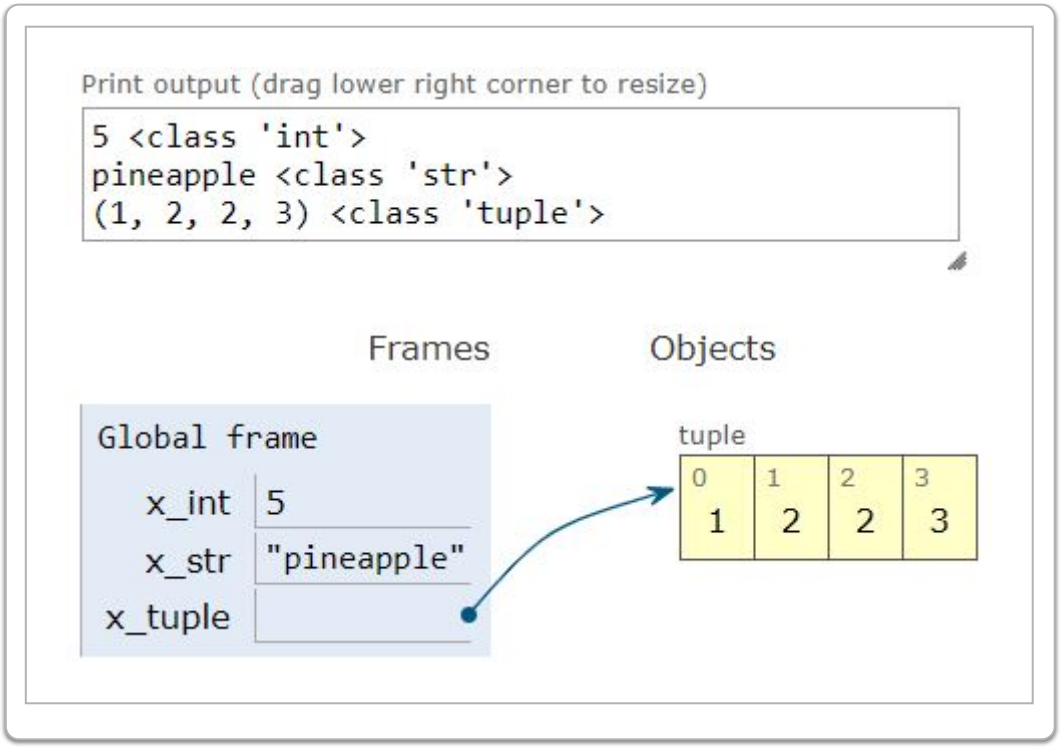


Sample Answer:

```
a = "seven"
b = 11
c = ("seven", 11)
d = {"tiger":"cat", "lion":"cat", "salmon":"fish"}
e = {11, 7}
f = 2j
g = 22
h = ["cat", "fish"]
```

type() Function

```
x_int = 5
x_str = 'pineapple'
x_tuple = (1, 2, 2, 3)
print(x_int, type(x_int))
print(x_str, type(x_str))
print(x_tuple, type(x_tuple))
```

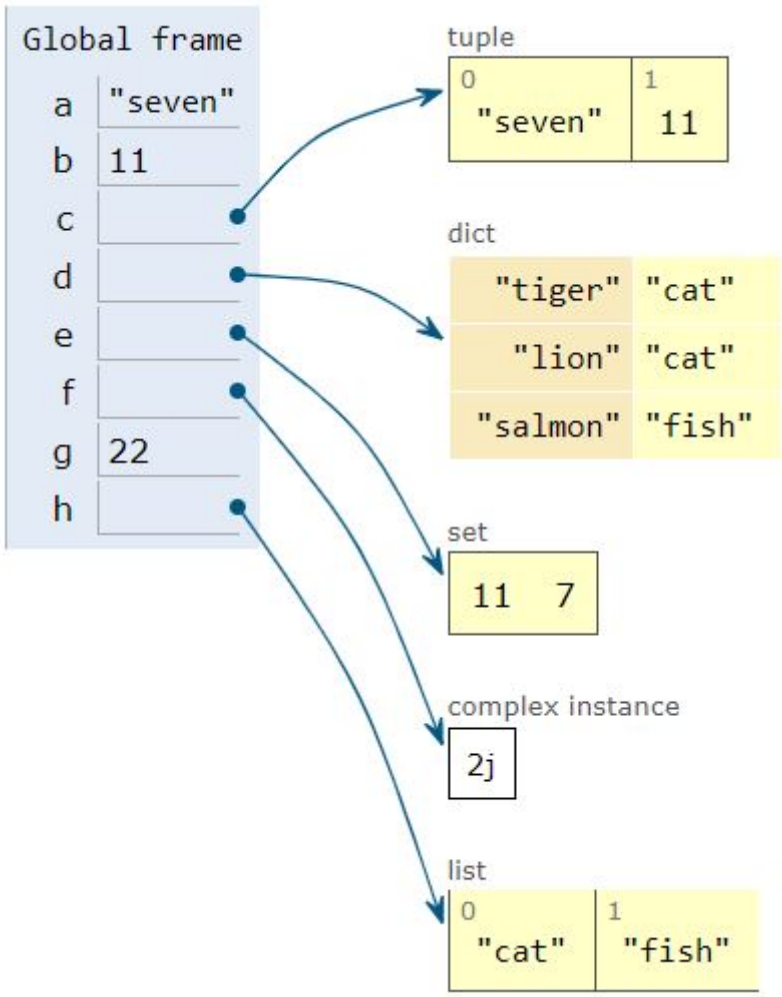


- *type*() is *polymorphic*

Test Yourself Exercises

Test Yourself: 2.1.03

Write code to print each type of object shown below.



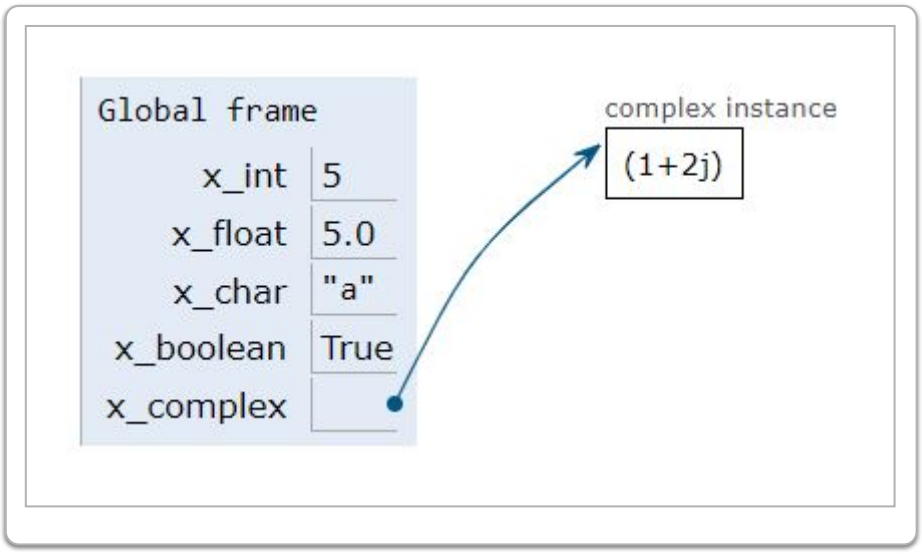
Sample Answer:

```
a = "seven"; b = 11; c = ("seven", 11)
d = {"tiger":"cat", "lion":"cat", "salmon":"fish"}
e = {11, 7}; f = 2j; g = 22
h = ["cat", "fish"]
objects_list = [a,b,c,d,e,f,g,h]
for next_object in objects_list:
    print(next_object, type(next_object))
```

```
seven <class 'str'>
11 <class 'int'>
('seven', 11) <class 'tuple'>
{'tiger': 'cat', 'lion': 'cat', 'salmon': 'fish'} <class 'dict'>
{11, 7} <class 'set'>
2j <class 'complex'>
22 <class 'int'>
['cat', 'fish'] <class 'list'>
```

Numeric Types

```
x_int = 5
x_float = 5.0
x_char = 'a'
x_boolean = True
x_complex = 1 + 2j # same as complex (1, 2)
```

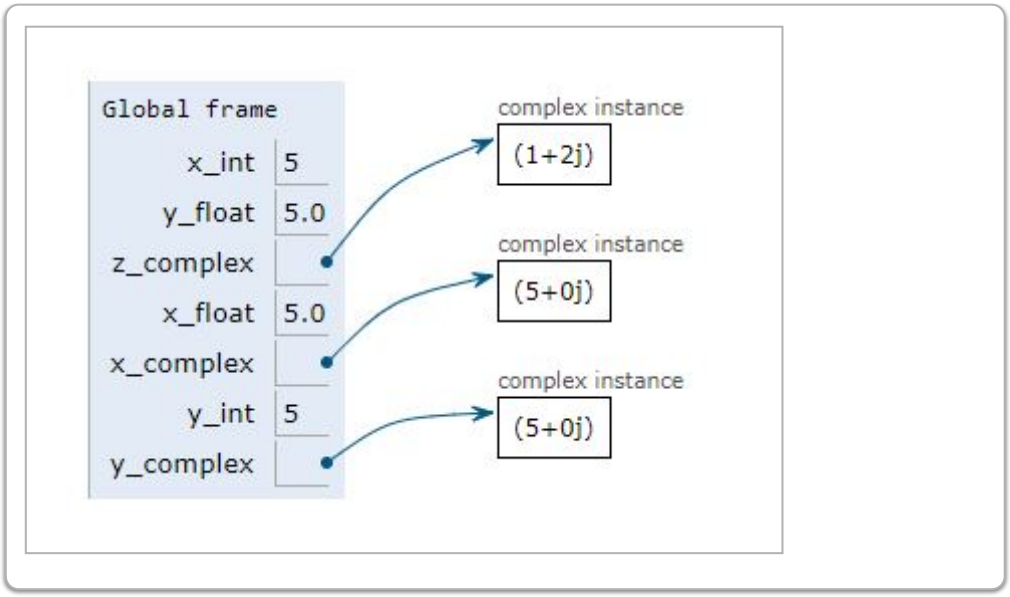


- three numeric types:

  integer, oat (real), complex

int(), *float*(), *complex*()

```
x_int = 5
y_float = 5.0
z_complex = 1 + 2j

x_float = float(x_int)
x_complex = complex(x_int)
y_int = int(y_float)
y_complex = complex(y_float)
```



- type "casting"

Type Casting

```
x_int = int(5); y_int = int(5.0); z_int = int("5")
x_float = float(5); y_float = float(5.0)
z_float = float('5')
x_str = str(5); y_str = str(5.0); z_str = str("5")
```

Integer Type

```
x_int = 5 # numeric literal
y_int = 999999999999999999999999
z_int = x_int + y_int
```

- unlimited length

- integers are objects with methods

Integer Representation

- different bases: 2, 8, 10, 16

```
x_int = 30  # default : base 10
x_bin = 0 b11110  # binary literal
x_oct = 0 o36     # octal literal
x_hex = 0 x1E     # hex literal
```

x_int \(= 3 \cdot 10^1 + 0 \cdot 10^0\)

x_bin \(= 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0\)

x_oct \(= 3 \cdot 8^1 + 6 \cdot 8^0\)

x_hex \(= 1 \cdot 16^1 + 14 \cdot 16^0\)

Integer Conversion

- different bases: 2, 8, 10, 16

```
x_int = 30 # default : base 10
x_bin = bin(x_int) # binary : base 2
x_oct = oct(x_int) # octal : base 8
x_hex = hex(x_int) # hex : base 16
```

x_int \(= 3 \cdot 10^1 + 0 \cdot 10^0\)

x_bin \(= 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0\)

x_oct \(= 3 \cdot 8^1 + 6 \cdot 8^0\)

x_hex \(= 1 \cdot 16^1 + 14 \cdot 16^0\)

Boston University Metropolitan College