DATA 200

SJSU

Python Interpreter

- Live Example

DATA 200

- Canvas
- Expectation
- Home Work
- Quizzes
- Lab
- Final Exam

Python Basics

- The ***Python interpreter*** is a computer program that executes code written in the Python programming language. An ***interactive interpreter*** is a program that allows the user to execute one line of code at a time.
- ***Code*** is a common word for the textual representation of a program (and hence programming is also called *coding*). A ***line*** is a row of text.
- The interactive interpreter displays a ***prompt*** (">>>") that indicates the interpreter is ready to accept code. The user types a line of Python code and presses the enter key to instruct the interpreter to execute the code. Initially you may think of the interactive interpreter as a powerful calculator.

Quiz

- What is the purpose of variable?
  - Store Value
  - Action statement for CPU
  - Automatically color the variable
- The code 30 * 60 is an expression?
  - True
  - False

Print Statement

- print() built in function
- print('hello World')
- Task print following:

Task1	Task2
Using 3 print statements, print following: Hi There What is your name? Sam?	Using 3 print statements, print following: Hello There. Your name is ...sam?

Print Statement

- Assume variable name = Sam, Age= 32, and City = “San Jose”.
- Is it correct?

```
    print(’Name’, name, 'years old.’, age, ‘city’,city)
```

Input()

- ***input()*** prebuilt function.
- The statement age= input() will read text entered by the user and assign the entered text to the age variable.
- The function input() causes the interpreter to wait until the user has entered some text and has pushed the return key.
- Question:

  Read the age from user and add 10 number to it? Do you what is the default return type of input()?

Variable and Assignment

- Which are correct from below for variable assignment?

Z=1
Z=y
X=y+2
X+1=3
X+Y=Y+X

Numeric Types: Floating Point

- A floating point number is real number like 98.6?
- The term "floating-point" refers to the decimal point being able to appear anywhere ("float") in the number.
- Scientific notation is useful for representing floating-point numbers that are much greater than or much less than 0, such as 6.02×10^{23}.
- Using ***scientific notation*** is written using an e preceding the power-of-10 exponent, as in 6.02e23 to represent 6.02×10^{23}

Numeric Types: Floating Point

- Convert to decimal
  - 1.0e-4
  - 7.2e-4

- Convert to Scientific Notation
  - 540000000
  - .000001

Division and Modulo

- 30/10 => 3.0
- 25//10 => ?
- The ***modulo operator*** (**%**) evaluates the remainder of the division of two integer operands. Ex: 23 % 10 is 3.

  78%2 => ?

  100 % (1 //2) =?

- Question:

  Given a 10-digit phone number stored as an integer, % and // can be used to get any part, such as the prefix. For phone_num = 6505562451 (whose prefix is 556)

Division and Modulo

Q Which gets the ten digit of x? if x = 693 which yields 9?

X % 10

X % 100

(x // 10) % 10

- Given a non-negative number x, which expression has the range -10 to 10? And what value of x we can take?

X % -10

(x % 21) - 10

(x % 20) - 1 0

Identifiers

- An ***identifier***, also called a ***name***, is a sequence of letters (a-z, A-Z), ***underscores*** (_), and digits (0–9), and must start with a letter
- Python is ***case sensitive***, meaning upper- and lowercase letters differ. Ex: "Cat" and "cat" are different. The following are valid names: c, cat, Cat, n1m1, short1, and _hello.

Errors

Options: TypeError, IndentiationError, NameError,SyntaxError,ValueError

Error	Expression
	lyric = 99 + " bottles of pop on the wall"
	print("Friday, Friday")
	int("Thursday")
	day_of_the_week = Friday
	print('Today is Monday")

Type Conversion

- An ***implicit conversion*** is a type conversion automatically made by the interpreter, usually between numeric types.

  - 1 + 2 =>
  - 1 + 2.0 => ?
  - 1.0 + 2.0 => ?

If-Else-Statement

```
if sales_type == 2:
        if sales_bonus < 5:
            sales_bonus = 10
        else:
            sales_bonus = sales_bonus + 2
else: sales_bonus = sales_bonus + 1
```

What would be the value of sales_bonus?

sales_type = 1; sales_bonus = 0;
sales_type = 2; sales_bonus = 4;
sales_type = 2; sales_bonus = 7;

If-Else-Statement

```
num_boxes = 0
num_apples = 9

if num_apples < 20:
    num_boxes = 3
if num_apples < 10:
    num_boxes = num_boxes - 1
```

If-Else-Statement

```
num_boxes = 0
num_apples = 9

if num_apples < 10:
    if num_apples < 5:
        num_boxes = 1
    else:
        num_boxes = 2
elif num_apples < 20:
        num_boxes = num_boxes + 1
```

If-Else-Statement

```
num_boxes = 0
num_apples = 9

if num_apples < 10:
    if num_apples < 5:
        num_boxes = 1
    else:
        num_boxes = 2
elif num_apples < 20:
        num_boxes = num_boxes + 1
```

String

A ***string*** is a sequence of characters, like the text SAM, that can be stored in a variable.

The string type is a special construct known as a ***sequence type***: A type that specifies a collection of objects ordered from left to right. A string's characters are ordered from the string's first letter to the last.

0	1	2
S	A	M

Q What character is in index 2 of the string "America"?

Q Write an expression that accesses the first character of the string my_country.

Assign my_var with the last character in my_str.

String

Q Is the this correct ?

```
name=‘sam’
name[0]=’p’
```

Q2: Is this correct?

```
string_1 = 'abc’
string_2 = '123’
concatenated_string = string_1 + string_2
print('Easy as ' + concatenated_string)
```

String

Q Is the this correct ?

```
name=‘sam’
name[0]=’p’
```

Q2: Is this correct?

```
string_1 = 'abc’
string_2 = '123’
concatenated_string = string_1 + string_2
print('Easy as ' + concatenated_string)
```

Conditional Expression

A ***conditional expression*** has the following form:

expr_when_true **if** condition **else** expr_when_false

Convert each if-else-statement to a single statement using conditional expression, using parenthesis around the condition

1)
```
If x < 100:
    y=0
else:
    y= x
```
2)
```
If x < 0:
    x = -x
else:
    x= x
```
3)
```
If x < 1:
    y = x
else:
    z =x
```

String Formatting

The format() function

- The string ***format()*** function allows a programmer to create a string with placeholders that are replaced by values or variable values at execution.
- A placeholder surrounded by curly braces { } is called a ***replacement field***. Values inside the format() parentheses are inserted into the replacement fields in the string.
- 'The {1} in the {0} is {2}.'.format('hat', 'cat', 'fat')
- 'The {animal} in the {headwear} is {shape}.'.format(animal='cat', headwear='hat', shape='fat')

String Formatting

Q: Determine the output of the following code snippets.

```
1.  print('April {}, {}'.format(22, 2020))
2.  date = 'April {}, {}'
    print(date.format(22, 2020))
3. date = 'April {}, {}'
   print(date.format(22, 2020))
   print(date.format(23, 2024))
4. print('{0}:{0}'.format(9, 43))
5. print('Hi {{{0}}}!'.format('Bilbo'))
6. month = 'April'
   day = 22
   print('Today is {month} {0}'.format(day, month=month))
```

Common Formatting Specs

Type	Description	Example	Output
s	String (default presentation type - can be omitted)	`'{:s}'.format('Aiden')`	`Aiden`
d	Decimal (integer values only)	`'{:d}'.format(4)`	`4`
b	Binary (integer values only)	`'{:b}'.format(4)`	`100`
x, X	Hexadecimal in lowercase (x) and uppercase (X) (integer values only)	`'{:x}'.format(15)`	`f`
e	Exponent notation	`'{:e}'.format(44)`	`4.400000e+01`
f	Fixed-point notation (6 places of precision)	`'{:f}'.format(4)`	`4.000000`
.[precision]f	Fixed-point notation (programmer-defined precision)	`'{:.2f}'.format(4)`	`4.00`

Common Formatting Specs

1. The value of num as a decimal (base 10) integer: `31`

```
num = 31
print('{:}'.format(num))
```

2. The value of num as a hexadecimal (base 16) integer: `1f`

```
num = 31
print('{:}'.format(num))
```

3. The value of num as a binary (base 2) integer: `11111`

```
num = 31
print('{:}'.format(num))
```

Common Formatting Specs

1. The value of num as a decimal (base 10) integer: `31`

```
num = 31
print('{:}'.format(num))
```

2. The value of num as a hexadecimal (base 16) integer: `1f`

```
num = 31
print('{:}'.format(num))
```

3. The value of num as a binary (base 2) integer: `11111`

```
num = 31
print('{:}'.format(num))
```

Order Of Evaluation

Q1: Which operator is evaluated first?

not y and x

and

not

Q2: Which operator has precedence?

w + 3 > x - y * z

+

-

>

*

Order Of Evaluation

Q1: In what order are the operators evaluated?

w + 3 != y - 1 and x

+, !=, -, and

+, -, and, !=

+, -, !=, and

Q2: To what does this expression evaluate, given x = 4, y = 7.

x == 3 or x + 1 > y

True

False

Order Of Evaluation

Q not x == 3 evaluates as not (x == 3).

Yes

No

Q w + x == y + z evaluates as (w + x) == (y + z).

Yes

No

Order Of Evaluation

Operator/Convention	Description	Explanation
( )	Items within parentheses are evaluated first	In `(a * (b + c)) - d`, the + is evaluated first, then *, then -.
* / % + -	Arithmetic operators (using their precedence rules; see earlier section)	`z - 45 * y < 53` evaluates * first, then -, then <.
`< <= > >= == !=`	Relational, (in)equality, and membership operators	`x < 2 or x >= 10` is evaluated as `(x < 2) or (x >= 10)` because < and >= have precedence over or.
not	not (logical NOT)	`not x or y` is evaluated as `(not x) or y`
and	Logical AND	`x == 5 or y == 10 and z != 10` is evaluated as `(x == 5) or ((y == 10) and (z != 10))` because and has precedence over or.
or	Logical OR	`x == 7 or x < 2` is evaluated as `(x == 7) or (x < 2)` because < and == have precedence over or

Common Types

Common data types.

Type	Notes
int	Numeric type: Used for variable-width integers.
float	Numeric type: Used for floating-point numbers.

Containers: sequence and mapping types.

Type	Notes
string	Sequence type: Used for text.
list	Sequence type: A mutable container with ordered elements.
tuple	Sequence type: An immutable container with ordered elements.
set	Set type: A mutable container with unordered and unique elements.
dict	Mapping type: A container with key-values associated elements.

Common Data Types

Q The list [‘2’,3,’b’] is invalid?

- True
- False

Q A sorted collection of integers might best be contained in list:

- Yes
- No

Q Student test scores that may later be adjusted, ordered from best to worst.

- List
- Tuple
- Dict

Q Student names and their current grades

- List
- Tuple
- Dict

Choosing Container type

Q Student test scores that may later be adjusted, ordered from best to worst.

- List
- Tuple
- Dict

Q Student names and their current grades

- List
- Tuple
- Dict

Q The final number of As, Bs, Cs, Ds, and Fs in the class

- List
- Tuple
- Dict

Finding Error Code

```
students = ['Jo', 'Bob', 'Amy']
grades = {} # Get student name, grade
name = input('name:')
grade = input('grade:') # Assign grade
grades.append(name) = grade
```

```
workers = ('Jo', 'Amy')
# Remove Amy from workers
del workers[1] # Print workers
print('Jo:', workers[0])
```

Write the Output

Write the simplest expression that captures the desired comparison.
1) x is a key in the dict my_dict

2) The variables x and y are unique objects.

3) The character 'G' exists in the string my_str

4) my_str is not the third element in the list my_list

Set Basics

Set basics

•A ***set*** is an unordered collection of unique elements. Sets have the following properties:

- •Elements are unordered: Elements in the set do not have a position or index.
- •Elements are unique: No elements in the set share the same value.

A set can be created using the ***set()*** function, which accepts a sequence-type iterable object (list, tuple, string, etc.)

whose elements are inserted into the set.

A ***set literal*** can be written using curly braces { } with commas separating set elements.

Note that an empty set can only be created using set().

Set Basics

```
# Initial list contains some duplicate values
first_names = [ ‘p', ‘s', ‘k', ‘j', 'Ma', 'Ro', ‘jon' ]

# Creating a set removes any duplicate values
names_set = set(first_names)

print(names_set)
```

Set Basics

Q What's the result of set(['A', 'Z'])?

- A set that contains 'A' and 'Z'.
- A list with the following elements: ['A', 'Z'].
- Error: invalid syntax.

Q What's the result of set(10, 20, 25)?

- A list with the following elements: [10, 20, 25].
- A set that contains 10, 20, and 25.

Q What's the result of set([100, 200, 100, 200, 300])?

- A list with the following elements: [100, 200, 100, 200, 300].
- A set that contains 100, 200, and 300.
- A set that contains 100, 200, 300, another 100, and another 200.
- Error: invalid syntax.

Set Basics

Sets are mutable – elements can be added or removed using set methods. The ***add()*** method places a new element into the set if the set does not contain an element with the provided value.
The ***remove()*** and ***pop()*** methods remove an element from the set.

sets support the `len()` function to return the number of elements in a set. To check if a specific value exists in a set, a membership test such as `value in set` (discussed in another section) can be used.

Adding elements to a set:

•set.add(value): Add value into the set. Ex: my_set.add('abc')
Remove elements from a set:

•set.remove(value): Remove the element with given value from the set. Raises KeyError if value is not found. Ex: my_set.remove('abc')
•my_set.pop(): Remove a random element from the set. Ex: my_set.pop()

Set Basics

Operation	Description
len(set)	Find the length (number of elements) of the set.
set1.update(set2)	Adds the elements in set2 to set1.
set.add(value)	Adds value into the set.
set.remove(value)	Removes value from the set. Raises KeyError if value is not found.
set.pop()	Removes a random element from the set.
set.clear()	Clears all elements from the set.

Set Basics

Assume that:

- monsters = {'Gorgon', 'Medusa'}
- trolls = {'William', 'Bert', 'Tom'}
- horde = {'Gorgon', 'Bert', 'Tom’}

Write the code to show:

```
{'Gorgon', 'Bert', 'Tom', 'Medusa', 'William’}
```

List Basics

A ***container*** is a construct used to group related values together and contains references to other objects instead of data. A ***list*** is a container created by surrounding a sequence of variables or literals with brackets [ ]. Ex: `my_list = [10, 'abc']` creates a new list variable my_list that contains the two items: 10 and 'abc'. A list item is called an ***element*** .

A list is also a sequence, meaning the contained elements are ordered by position in the list, known as the element's ***index***, starting with 0. `my_list = [ ]`. creates an empty list.

Exercise

Q Write a statement that performs the desired action.

Assume the list house_prices = ['$140,000', '$550,000', '$480,000'] exists.

1. Update the price of the second item in house_prices to '$175,000’.
2. Add a price to the end of the list with a value of '$1,000,000’.
3. Remove the 1st element from house_prices, using the pop() method.
4. Remove '$140,000' from house_prices, using the remove() method.

Sequence-type methods and functions

Operation	Description
len(list)	Find the length of the list.
list1 + list2	Produce a new list by concatenating list2 to the end of list1.
min(list)	Find the element in list with the smallest value.
max(list)	Find the element in list with the largest value.
sum(list)	Find the sum of all elements of a list (numbers only).
list.index(val)	Find the index of the first element in list whose value matches val.
list.count(val)	Count the number of occurrences of the value val in list.

Dictionary Basics

1. A ***dictionary*** is a Python container used to describe associative relationships. A dictionary is represented by the ***dict*** object type. A dictionary associates (or "maps") keys with values. A ***key*** is a term that can be located in a dictionary, such as the word "cat" in the English dictionary
2. . A ***value*** describes some data associated with a key, such as a definition. A key can be any immutable type, such as a number, string, or tuple; a value can be any type.
3. A dict object is created using ***curly braces*** { } to surround the ***key:value pairs*** that comprise the dictionary contents. Ex: `players = {'Lionel Messi': 10, 'Cristiano Ronaldo': 7}` creates a dictionary called players with two keys: 'Lionel Messi' and 'Cristiano Ronaldo', associated with the values 10 and 7 (their respective jersey numbers). An empty dictionary is created with the expression `players = { }`.
4. Dictionaries are typically used in place of lists when an associative relationship exists. Ex: If a program contains a collection of anonymous student test scores, those scores should be stored in a list. However, if each score is associated with a student name, a dictionary could be used to associate student names to their score.

Dictionary Basics

Q: Use braces to create a dictionary called ages that maps the names 'Bob' and 'Frank' to their ages, 27 and 75, respectively. For this exercise, make 'Bob' the first entry in the dict.

Q Is it correct statement?

```
prices = {'apples': 1.99, 'oranges': 1.49}
print(f'The price of apples is {prices["apples"]}')
print(f'\nThe price of lemons is {prices["lemons"]}')
```

Q A dictionary entry is accessed by placing a key in curly braces { }.

True

False

Q Dictionary entries are ordered by position.

True

False

Modifying Dictionary

1) Which statement adds 'pears' to the following dictionary?

```
prices = {'apples': 1.99, 'oranges': 1.49, 'kiwi': 0.79}
```

   - ○ prices['pears'] = 1.79
   - ○ prices['pears': 1.79]

2) Executing the following statements produces a KeyError:

```
prices = {'apples': 1.99, 'oranges': 1.49, 'kiwi': 0.79}
del prices['limes']
```

   - ○ True
   - ○ False

3) Executing the following statements adds a new entry to the dictionary:

```
prices = {'apples': 1.99, 'oranges': 1.49, 'kiwi': 0.79}
prices['oranges'] = 1.29
```

   - ○ True
   - ○ False

While Loop

```
while expression: # Loop expression # Loop body: Sub-statements to execute # if the loop expression evaluates to True # Statements to execute after the expression evaluates to False
```

How many times will loop body execute?

```
x = 3
while x >= 1: # Do something
  x = x – 1
```

Assume user would enter 'n', then 'n', then 'y'.

```
# Get character from user here
while user_char != 'n': # Do something # Get character from user here
```

Assume user would enter 'a', then 'b', then 'n'.

```
# Get character from user here
while user_char != 'n': # Do something # Get character from user here
```

While Loop

Complete the loop expressions, using a single operator in your expression. Use the most straightforward translation of English to an expression.

Iterate while x is less than 100.
Iterate while x is greater than or equal to 0.
Iterate while c equals 'g’.
Iterate *until* c equals 'z’.

While Loop

What is the output of

```
x = 5
y = 18
while y >= x:
    print(y, end=' ')
    y = y - x
```

```
x = 1
y = 3
z = 5
while not (y < x < z):
    print(x, end=' ')
    x = x + 1
```

1) What sequence is generated by `range(7)`?

- ○ 0 1 2 3 4 5 6 7
- ○ 1 2 3 4 5 6
- ◉ 0 1 2 3 4 5 6

2) What sequence is generated by `range(2, 5)`?

- ◉ 2 3 4
- ○ 2 3 4 5
- ○ 0 1 2 3 4

Range Function

Using the range() function.

Range	Generated sequence	Explanation
`range(5)`	`0 1 2 3 4`	Every integer from 0 to 4.
`range(0, 5)`	`0 1 2 3 4`	Every integer from 0 to 4.
`range(3, 7)`	`3 4 5 6`	Every integer from 3 to 6.
`range(10, 13)`	`10 11 12`	Every integer from 10 to 12.
`range(0, 5, 1)`	`0 1 2 3 4`	Every 1 integer from 0 to 4.
`range(0, 5, 2)`	`0 2 4`	Every 2nd integer from 0 to 4.
`range(5, 0, -1)`	`5 4 3 2 1`	Every 1 integer from 5 down to 1
`range(5, 0, -2)`	`5 3 1`	Every 2nd integer from 5 down to 1

Exercise

Write the simplest range() function that generates the appropriate sequence of integers.

- Every integer from 0 to 500.
- Every integer from 10 to 20
- Every 2nd integer from 10 to 20
- Every integer from 5 down to -5

Exercise

You need to find solutions to a set of equations.
Ex: 8x + 7y = 38 and 3x - 5y = -1 have a solution x = 3, y = 2.

Given integer coefficients of two linear equations with variables x and y, use brute force to find an integer solution for x and y in the range -10 to 10.

Assume user input following:

```
8
7
38
3
-5
-1

Ouput

3 2
```

Exercise

Write a program that reads a list of integers into a list as long as the integers are greater than zero, then outputs the smallest and largest integers in the list.

If the input is:

10 5 3 21 2 -6

The output is 2 21