Midterm 2 Preparation

Python

Thomas Schwarz, SJ, Marquette University

Controlling Output

- Controlling print statements is necessary for good-looking terminal output
  - Two avenues:
    1. Resetting default parameters in print
    2. Using the format statement to create good-looking strings

Controlling Output

- Default parameters in print
  - sep : The separator between different arguments

```
>>> print(1, 2, 3, 4, sep='; ')
1; 2; 3; 4
```

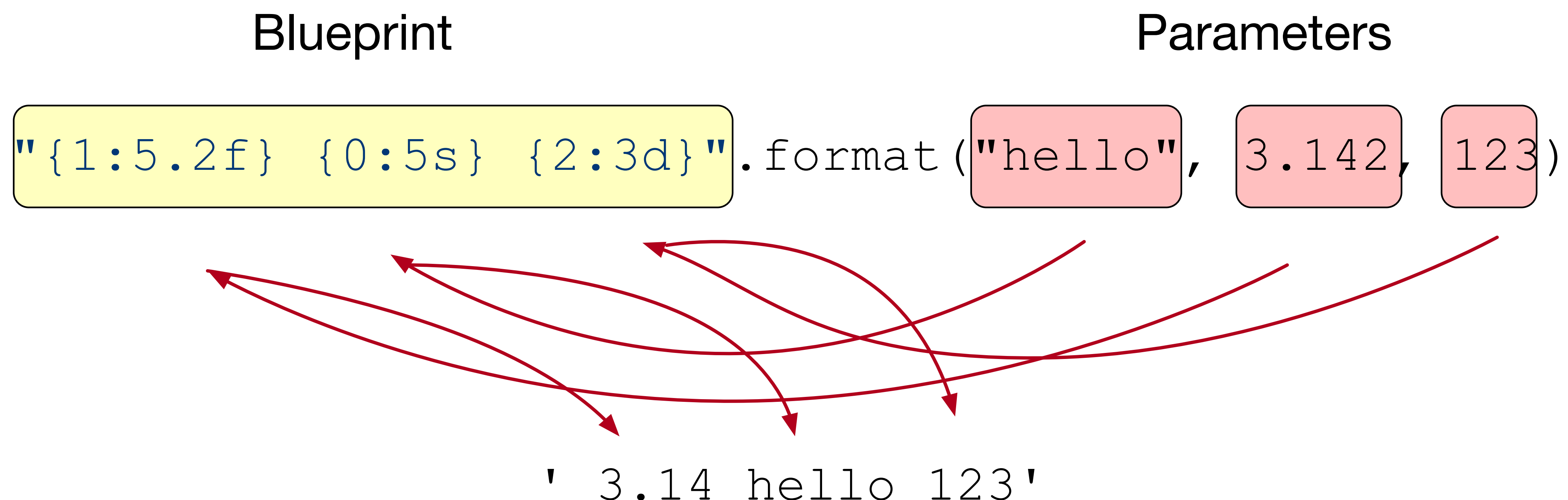
  - end: The terminating string

```
>>> print(1, 2, 3, 4, end="\n------------------\n")
1 2 3 4
------------------
```

  - file: The file to be written to. Default is standard I/O
  - flush: If set to True, write immediately

Controlling Output

- The format statement allows us to compose strings
- Consists of a blueprint string to which the format method is applied.
    - The insertion of the parameters is controlled by the contents of the curly bracket.



Controlling Output

- The curly brackets indicate places where data gets inserted into the string
  - If they are left empty, then parameters get filled in in order
  - Otherwise, if they contain just numbers, the numbers specify the coordinate in the arguments tuple

```
>>> "{1} {0} {2}".format("hello", 3.142, 123)
'3.142 hello 123'
```

Controlling Output

- After a colon, we can specify how the argument is interpreted
  - Types are:
    - s — string
    - e, E, f, F, g, G — floating point in exponential or fixed point format; g means general and switches between exponential and fixed
    - d, n — integer
    - b, o, x — integer converted to binary, octal, or hexadecimal format
    - c - character: integer is converted to unicode
    - n - number with separation according to locale

Controlling output

- Before the type specifier, we can give size of the field
  - 10s — ten characters
  - 6.2f — six digit fixed point number with two digits after the point
- We can also specify the alignment:
  - < — left, > — right, ^ — center

Controlling output

- Nice examples:
    - We can use the percentage sign inside the brackets to specify percentages

```
>>> "The percentage is {:.2}%.".format(45/57)
'The percentage is 0.79%.'
>>> "The percentage is {:.2%}.".format(45/57)
'The percentage is 78.95%.'
```

Controlling output

- Nice examples:
  - We can specify the filler

```
>>> "{:*^30}".format("centered")
'***********centered***********'
>>> "{:*<30}".format("left aligned")
'left aligned******************'
>>> "{:*>30}".format("right aligned")
'*****************right aligned'
```

Controlling output

- In Python 3.6 and later, you can use fstrings.
  - The syntax is simpler
  - Put an f or F before the beginning quotation mark

```
>>> name = "Cleese"
>>> first_name = "John"
>>> f"Monty Python member {first_name} {name} is funny."
'Monty Python member John Cleese is funny.'
```

Controlling Loops

- Python has two statements to control behavior within a loop
    - `continue` — stops the execution of the current loop and starts the next loop
    - `break` — stops the execution of the loop completely

Controlling Loops

- Create a list of random numbers 1/r with -10 <= r <= 10:
    - If the random number is zero, we just go to the next iteration

```
import random

def create_random_inverses(number):
    result = [ ]
    while len(result)<number:
        r = random.randint(-10, 10)
        if r==0:
            continue
        result.append(1/r)
    return result

if __name__ == "__main__":
    print(create_random_inverses(50))
```

Controlling Loops

- Trying to find a number such that f(x) is close to 0.
  - Warning: This is not a good way to solve an equation.
  - It’s like hunting deer by just shooting in the dark.
  - People might get hurt! Deers however are usually safe.

```
def f(x):
    return math.sin(x)**3+
       math.log(x,2)/math.exp(x-1)

def solve(f, a, b):
    while True:
        guess = random.uniform(a,b)
        if abs(f(guess)-0) < 0.001:
            break
    print( f"{guess} is now close to
           being a solution" )

if __name__ == "__main__":
    solve(f, 0, 11)
```

Lists, Dictionaries, Tuples, Sets,

- You are given a string. Return the same string with all white spaces removed.

Solution

1. Use a for loop.

```
def remove_white_spaces(string):
    result = []
    for letter in string:
        if letter not in " \t\n":
            result.append(letter)
    return "".join(result)
```

Empty list for the result.

Walk through string.

Select which letters to append

Return result list as a string

Solution

2. Use list comprehension.

```
def remove_white_spaces_c(string):
    result = [c for c in string if c not in " \t\n"]
    return “".join(result)
```

Lists, Dictionaries, Tuples, Sets,

- You are given two strings. You can assume that they have the same length. Create a dictionary that associates the first character in string 1 to the first character in string 2, the second character in string 1 to the second character in string 2, … Previous associations might be overwritten
- Example:
    - “apple”, “banana” —> {‘a’: ‘b’, ‘p’: ’n’, ‘l’: ‘a’, ‘e’: ’n’}
    - ‘p’ was associated first with ‘a’, but then the association changed to ’n'

Solution

1. Use a for loop over the indices

```
def associate(string1, string2):
    dictionary = {}
    for i in range(min(len(string1), len(string2))):
        dictionary[string1[i]]=string2[i]
    return dictionary
```

Make sure to avoid an index error

Solution

- Or use dictionary comprehension and zip

```
def associate_c(string1, string2):
    return {key:value for key, value in zip(list(string1), list(string2))}
```

Convert strings into lists

zip to make a list of tuples

tuple extraction

dictionary comprehension

Solution

- Or even simpler, let Python do the dirty work:
  - zip works on iterables like strings, not only on lists
  - Keyword `dict` makes a dictionary out of a list of pairs

```
def associate_z(string1, string2):
    return dict(zip(string1, string2))
```

Lists, Dictionaries, Tuples, Sets,

- You are given a translation dictionary with letters for keys and values.
- Write a function that substitutes the letters in a string according to the dictionary.
    - Example: {‘a’: ‘1’, ‘e’: ‘2’, ‘i’: ‘3’, ‘o’: ‘4’, ‘u’: ‘5’}
    - “thomas schwarz” —> “th4m1s schw1rz”

Solution

- Use a for loop, aggregating the new string as a list of characters

```
def translate(string, dictionary):
    result = []
    for letter in string:
        if letter in dictionary:
            result.append(dictionary[letter])
        else:
            result.append(letter)
    return "".join(result)
```

Solution

- Or use a ternary operator
  - `value1 if cond_is_true else value2`
- Expression is value1 if the condition is true, otherwise it is value 2
- Then we can use list comprehension

```
def translate_c(string, dictionary):
    return "".join([dictionary[letter] if letter in dictionary
                    else letter for letter in string]
```