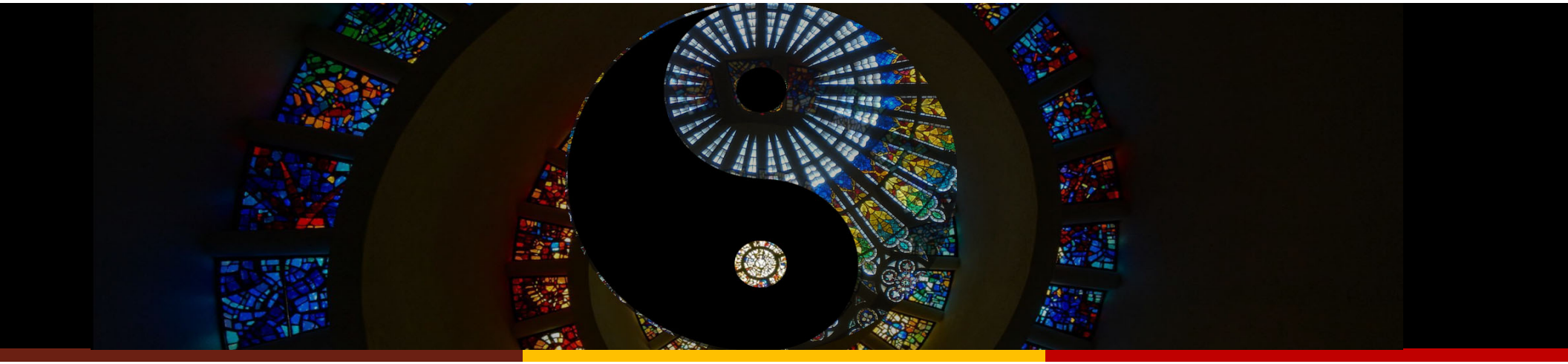


Python Dictionary and Counting

Logical Thinking of Informatics

Lab 4

Thinking Processes

Algorithm Discovery

- The development of a program consists:
  - Discovering the underlying algorithm
  - Representing that algorithm as a program

- Theory of problem solving
  - The algorithm to generate an algorithm for any particular problem is purely imaginary
  - There are certain problems that are **unsolvable**!!
  - The ability to solve problems is more like an **artistic skill** to be developed
    - Practices

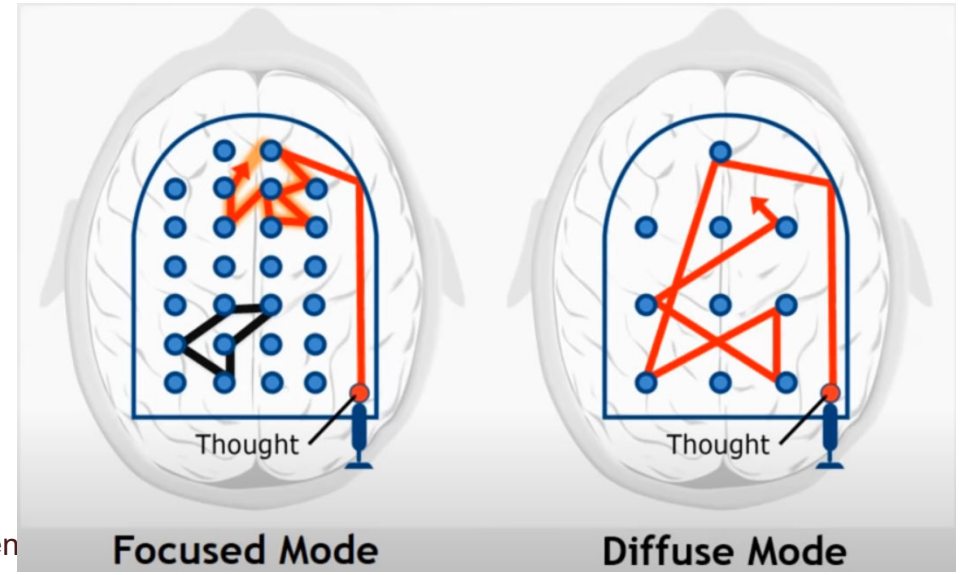
Problem Solving Phases

1. Understand the problem
2. Get an idea how an algorithmic procedure might solve the problem.
3. Formulate the algorithm and represent it as a program
4. Evaluate the program for accuracy and its potential as a tool for solving other problems

⇒Not necessarily completed in sequence

Incubation Periods

- Between conscious work and the sudden inspiration
  - Reflect a process
    - A subconscious part of the mind appears to continue working
    - Forces the solution into the conscious mind
  - Various for different people



Logical Thinking

- Break down the big problem (logically)
  - Divide
- Combine the smaller puzzles (logically)
  - Conquer

Algorithm Primitives and Structures

- Primitives
  - Assignment — *name* ← *expression*
  - Conditional selection — **if** *condition* **then** *action*
  - Repeated execution — **for** *condition* **do** *activity*
  - Procedure — **procedure** *name* (*generic names*)

- Repetitive structures used in describing algorithmic processes
  - Iterative structures
  - Recursive structures

Algorithm Primitives and Structures

- Primitives
  - Assignment — *name* ← *expression*
  - Conditional selection — **if** *condition* **then** *action*
  - Repeated execution — for condition do activity
  - Procedure — **procedure** *name* (*generic nmes*)

```
myList=[]
myList2=["test",3,18.0,"hello"]
```

Algorithm Primitives and Structures

- Primitives
  - Assignment — *name* ← *expression*
  - Conditional selection — **if** *condition* **then** *action*
  - Repeated execution — for condition do activity
  - Procedure — **procedure** *name* (*generic names*)

```
from datetime import date
today=date.today()
if (today=date(2023,4,10)):
  print("we will have Lab3 class")
else:
  print("No Lab Class!")
```

Algorithm Primitives and Structures

- Primitives
  - Assignment — *name* ← *expression*
  - Conditional selection — **if** *condition* **then** *action*
  - Repeated execution — **for condition do activity**
  - Procedure — **procedure** *name* (*generic names*)

```
skip=1
boundary=[0,100]
myList=[]
for value in range(boundary[0],boundary[1],skip):
    myList.append(value)

print(myList)
```

Algorithm Primitives and Structures

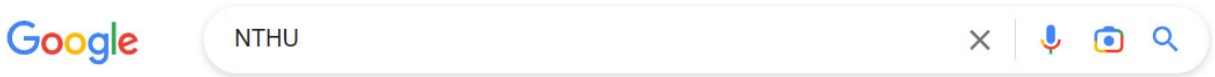
- Primitives
  - Assignment — *name* ← *expression*
  - Conditional selection — **if** *condition* **then** *action*
  - Repeated execution — for condition do activity
  - Procedure — **procedure** *name* (*generic names*)

```
def genIntList(low,high,skip, exclude):
  myList=[]
  for value in range(low,high,skip):
    if ((value%exclude)!=0):
      myList.append(value)
  return(myList)
```

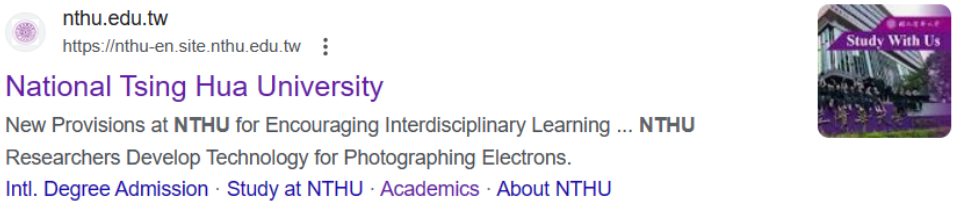
Dictionary

Dictionary

- Dictionaries are used to store data values in key:value pairs
- Key
  - Key is an element in a set
  - Key can represent a record
- Key:value can be related to the design of associative memory in the 1960s
  - Key: some kind of address

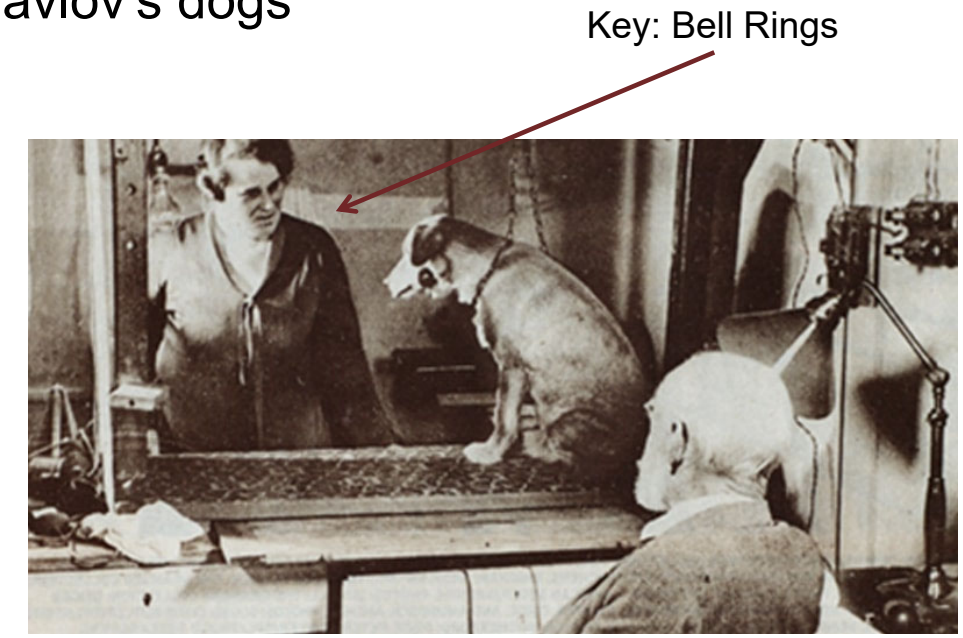


  - Value: the block of memory/records related to the key



Associative Memory

- Example: Pavlov's dogs



Value: Memory

Value: Salivation

Dictionary

- Dictionaries are used to store data values in key:value pairs
- A dictionary is a collection which is
  - ordered*
  - Changeable
  - Not allow duplicates
- Example

key value

```
thisDic={"Apple":2, "Orange": 15, "Year":2021}
print(thisDic)
```

```
{'Apple': 2, 'Orange': 15, 'Year': 2021}
```

```
print("The year for this Dictionary is: %d" % (thisDic["Year"]))
```

```
The year for this Dictionary is: 2021
```

Practice Possible Answers

- When you have a list of student names and corresponding grades, which data structure is more suitable for storing this information?

```
student_grade={"Alice":80, "Bob":85, "Charles": 95}
print(student_grade)
print(student_grade["Alice"])
```

```
{'Alice': 80, 'Bob': 85, 'Charles': 95}
80
```

Practice Possible Answers

- If you want to store a collection of random words without any associations, which data structure would you use?

```
words = ["apple", "dog", "umbrella", "moon", "pencil"]
print(words[3])
```

```
moon
```

Practice Possible Answers

- When you want to store data about a product, such as its name, price, and quantity, which data structure is better suited?

```
product = {"name": "Laptop", "price": 999, "quantity": 10}
print(product["price"])
```

```
999
```

What If Multiple Products?

- Can have a list of dictionaries

```
products = [
    {"name": "Laptop", "price": 999, "quantity": 10},
    {"name": "Smartphone", "price": 799, "quantity": 15},
    {"name": "Tablet", "price": 499, "quantity": 8}
]
```

```
for product in products:
    print("Product Name:", product["name"])
    print("Product Price:", product["price"])
    print("Product Quantity:", product["quantity"])
    print()
```

Why Dictionary?

Advantages of Dictionary

- Key-value pairs
- Unique keys
- Constant-time lookups
- Flexible keys
- Nested data structures

Constant Look Up

- Calculate the total cost of purchasing a specific quantity of each fruit. Use the following data: apples cost $2, bananas cost $1, and oranges cost $1.5. Purchase 5 apples, 3 bananas, and 7 oranges.

```
# Create a dictionary with fruit prices
fruit_prices = { "apple": 2, "banana": 1, "orange": 1.5 }

# Define the quantities of each fruit
quantities = {   "apple": 5, "banana": 3, "orange": 7}
```

```
# Calculate the total cost
total_cost =  (fruit_prices["apple"] * quantities["apple"] +
 fruit_prices["banana"] * quantities["banana"] +
 fruit_prices["orange"] * quantities["orange"])
```

```
# Print the total cost
print("Total cost: $ %.2f" % (total_cost))
```

```
Total cost: $ 23.50
```

Counting a Long List

- How to count the occurrences of a long list of numbers separated by commas

```
longInput="0,1,3,5,1,5,3,6,8,1,4,8,2,1,3,2,5,2,4"
```

Counting a Long List

```
longInput="0,1,3,5,1,5,3,6,8,1,4,8,2,1,3,2,5,2,4"
```

```
def readStringToDict(inputStr,splitSymbol):
  inList=inputStr.split(splitSymbol)
  resultDict={}
  for element in inList:
    if (element in resultDict.keys()):
      resultDict[element]=resultDict[element]+1
    else:
      resultDict[element]=1
  return(resultDict)
```

```
inputDic=readStringToDict(longInput,",")
print(inputDic)
```

```
{'0': 1, '1': 4, '3': 3, '5': 3, '6': 1, '8': 2, '4': 2, '2': 3}
```