Statements in Python

Algorithm

- Well-defined instructions to carry out a particular task
- It is predictable, deterministic, and not subject to chance
- Algorithm is like giving driving directions
- It consists of logical sequence of operations
- Written out using pseudo code

What makes up a Python script?

Variables, and Data structures

Built-in functions

Assignment statements

Input/Output statements

Program control statements

Blocks for reusing code - def, class, modules

Python coding hierarchy

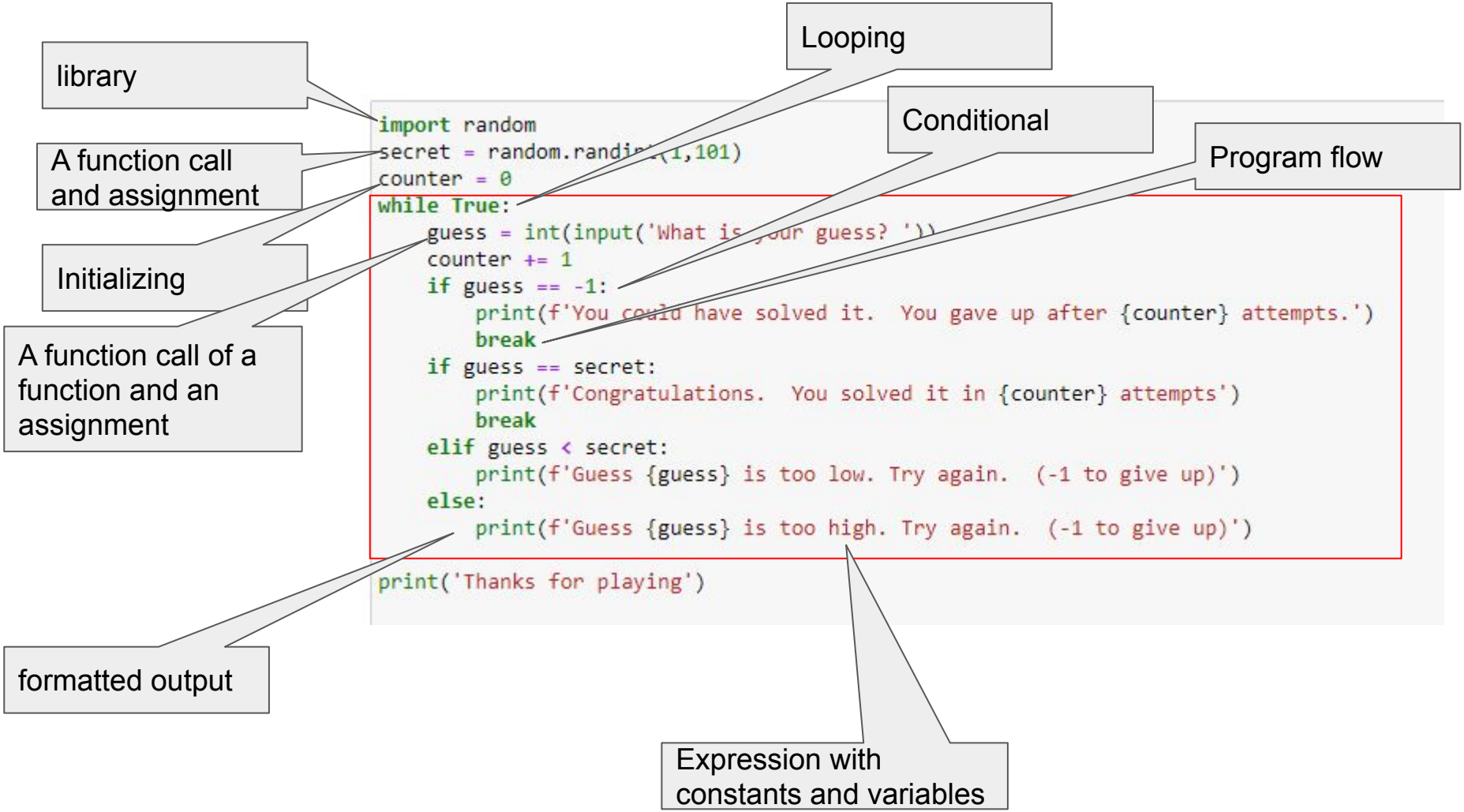
- Programs saved as .py files; they are called modules
  - Modules contain statements
  - Statements are executed sequentially
  - Types of statements: Assignment, conditional, looping, program flow, def, class
    - Statements contain expressions
      - Expressions are created using constants, variables, values from functions, values from attributes and operators
      - Expressions can use variables and process the variables

User interaction

Use pseudocode to write the logic

I'm thinking of a number between 1 and 100. Can you guess it?

Think up a random number between 1 and 100 and store it to secret
Ask user to guess it (or -1 to give up)
    if the guess matches secret, congratulate and end the game
    if the guess smaller than secret, say 'guess is too low', counter++, continue the game
    if the guess is larger than secret, say 'guess is too high' counter++, continue the game


library
A function call and assignment
Initializing
A function call of a function and an assignment
formatted output
Looping
Conditional
Program flow
Expression with constants and variables
import random
secret = random.randint(1,101)
counter = 0
while True:
guess = int(input('What is your guess? '))
counter += 1
if guess == -1:
print(f'You could have solved it. You gave up after {counter} attempts.')
break
if guess == secret:
print(f'Congratulations. You solved it in {counter} attempts')
break
elif guess < secret:
print(f'Guess {guess} is too low. Try again. (-1 to give up)')
else:
print(f'Guess {guess} is too high. Try again. (-1 to give up)')
print('Thanks for playing')

Python keywords

and	as	assert	async	await	break	class
continue	def	del	elif	else	except	False
finally	for	from	global	if	import	in
is	lambda	None	nonlocal	not	or	pass
raise	return	True	try	while	with	yield

Modify this code to allow multiple plays

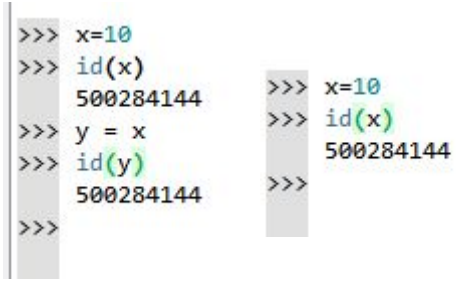
```
import random
secret = random.randint(1,101)
counter = 0
while True:
    guess = int(input('What is your guess? '))
    counter += 1
    if guess == -1:
        print(f'You could have solved it.  You gave up after {counter} attempts.')
        break
    if guess == secret:
        print(f'Congratulations.  You solved it in {counter} attempts')
        break
    elif guess < secret:
        print(f'Guess {guess} is too low. Try again.  (-1 to give up)')
    else:
        print(f'Guess {guess} is too high. Try again.  (-1 to give up)')

print('Thanks for playing')
```

Somewhat unique Python characteristics

- curly braces are { } are not used to mark begin and end of blocks
- Compound statements - header line ends with a : and following lines are indented
- parentheses are optional - like in the if statement
- do not use ; to mark end of line. Use it to separate multiple statements on a single line

Assignment statements

Assignment statements



- Assignments are fundamentals to programming.
- *variableName = expression*
- Read as "expression assigned to variableName", not variableName equals expression
- Variable names: Begins with an underscore or an alphabet; case sensitive; can't be a reserved word
- This creates an object reference
- Each object gets an *id*
- When you assign an object another object it gets the same id

Names are created when first assigned

Do not have to *declare* variable data types

Study these assignment statements

x is created

then *x* is assigned to *y*

x, and *y* have the same *id*

then *x* is incremented - creating a new *x*

and new object *id* created

```
>>> x=10
>>> id(x)
    500284144
>>> y = x
>>> id(y)
    500284144
>>> x = x+1
>>> id(x)
    500284176
>>> y
    10
>>> x
    11
>>>
```

More on Python assignment statements

unpacking assignments; assign many values to many variables with one statement

- a,b = 1,2 #positional assignments. What value does b get?
- s = 'spam'
- a,b,c,d = 'spam' #unpacks and assigns

Multiple target assignments

- x = y = 4 #what value does x get?

More on Python assignments

Augmented assignment

- x = 10
- x += 4
- x *= 4 # x*4 is assigned to x
- s = 'spam'
- s += 'eggs'

More on Python assignments

swapping variables

```
x = 1

y = 2

x, y

x, y = y, x

x, y
```

if statement

Basic *if* statement

```
grade=85
if grade >= 60:
  print('passed')
else:
  print('failed')

#Boolean expression evaluates to True or False
```

Boolean expressions

- True
- bool(1)
- bool(0)
- bool(-1)
- bool(2.3)
- bool('abc')
- bool('')
- 2 or 3
- 3 or 2
- 2 and 3
- bool(None)

- x = 1
- y = 1.0
- x == y
- x != y
- x >y
- x>=y
- x<=y
- x<y

See results in this colab doc

if statement example: create vampire.py

```
name = input('What is your name?')
age  = int(input('What is your age?'))
if name == 'Alice':
    print('Hi, Alice.')
elif age < 12:
    print('You are not Alice, kiddo.')
elif age > 2000:
    print('Unlike you, Alice is not an undead, immortal vampire.')
elif age > 100:
    print('You are not Alice, grannie.')
else:
    print('You are not over the hill')
```

Know your future

#Magic ball

```
import random
def getAnswer(answerNumber):
    if answerNumber == 1:
       return 'It is certain'
    elif answerNumber == 2:
        return 'It is decidely decidedly so'
    elif answerNumber == 3:
        return 'Yes'
    elif answerNumber == 4:
        return 'Reply hazy try again'
    elif answerNumber == 5:
        return 'Ask again later'
    elif answerNumber == 6:
        return 'Concentrate and ask again'
    elif answerNumber == 7:
        return 'My reply is no'
    elif answerNumber == 8:
        return 'Outlook not so good'
    elif answerNumber == 9:
        return 'Very doubtful'
question=  input('What do you want to know about the future?')
```

Future 2.0

#Magic ball 2

```
import random

messages = ['It is certain',
    'It is decidedly so',
    'Yes definitely',
    'Reply hazy try again',
    'Ask again later',
    'Concentrate and ask again',
    'My reply is no',
    'Outlook not so good',
    'Very doubtful']

question=  input('What do you want to know about the future?')
print(messages[random.randint(0, len(messages) - 1)])
```

Conditional assignment

Python has a conditional assignment feature

```
x = 15

y = 100 if x>=10 else 200

# y will be assigned 200
```

while statement

while statement

Allows you to repeat one or more actions while a condition remains true

```
product = 3

while product <50:

  product = product * 3

  print(product)

print('Exited the loop')
```

-

The while true: loop pattern is useful for setting up interactive user interfaces.

```
while True:
    print('Who are you?')
    name = input()
    if name != 'Joe':
        continue
    print('Hello, Joe. What is the password? (It is a fish.)')
    password = input()
    if password == 'swordfish':
        break
print('Access granted.')
```

#validate inputs

```
while True:
    print('Enter your age:')
    age = input()
    if age.isdecimal(): // string method
        break
    print('Please enter a number for your age.')
```

#validate inputs

```
while True:
        print('Select a new password (letters and numbers only):')
        password = input()
        if password.isalnum():
            break
        print('Passwords can only have letters and numbers.')
```

#Exit example

```
while True:
    print('Type exit to exit.')
    response = input()
    if response == 'exit':
        break
    print('You typed ' + response + '.')
```

Too many cats

#My cats - version 1

```
print('Enter the name of cat 1:')
catName1 = input()
print('Enter the name of cat 2:')
catName2 = input()
print('Enter the name of cat 3:')
catName3 = input()
print('Enter the name of cat 4:')
catName4 = input()
print('Enter the name of cat 5:')
catName5 = input()
print('Enter the name of cat 6:')
catName6 = input()
print('The cat names are:')
print(catName1 + ' ' + catName2 + ' ' + catName3 + ' ' + catName4 + ' ' + catName5 + ' ' +
catName6)
```

Too many cats

#Mycats - Version 2

```
catNames = [] #empty list
while True:
    print('Enter the name of cat ' + str(len(catNames) + 1) + ' (Or enter nothing to stop.):')
    name = input()
    if name == '':
        break
    catNames = catNames + [name]  # list concatenation
print('The cat names are:')
for name in catNames:
    print('  ' + name)
```

for statement

for statement

Repeat an action or actions a number of times the number of times can be predetermined or dynamic

for statement

If you wish to run statements a fixed number of times, use range(low, high)

```
x = range(0,10)

for i in x:

  print (i, i**2)
```

using range function

```
# create even numbers between 0 and 9

for i in range(0,10,2):

  print(i)
```

for statement

you can iterate through the characters in a string using the for loop. Run this code to see the result

```
string1 = "abcdefghijkl"
i=0
for c in string1:
  print (c,i)
  i += 1
```

for statement

You can iterate through items in a list using a for statement

```
fruits = ['apples', 'grapes', 'plums', 'mangos']
for fruit in fruits:
    print(fruit.upper())
```

for use in simulation

#simulating Coin flips

```
import random
heads = 0
for i in range(1, 1001):
    if random.randint(0, 1) == 1:
        heads = heads + 1
    if i == 500:
        print('Halfway done!')
print('Heads came up ' + str(heads) + ' times.')
```

Algorithm development from requirements

Requirements state what need to be done, and not how it can be done. Can you figure the logic these requirements? Create task1.py, task2.py, task3.py and task4.py

- Task 1: Take 10 integer inputs from the user; print the sum of those numbers
- Task 2: Take 10 integer inputs from the user; print the average of those numbers
- Task 3: Take 10 integer inputs from the user; find the largest of those numbers
- Task 4: Take 10 integer inputs from the user; find the smallest of those numbers

Formatting strings

```
number1 = 10.25

print(f'number1 {number1}') # no format

print(f'number1 {number1:6.2f}') # 2 decimals

print(f'number1 {number1:<6.2f}') # 2 decimals, aligned left

print(f'number1 {number1:>6.2f}') # 2 decimals, aligned right

print(f'number1 {number1:^6.2f}') # 2 decimals, aligned center

print(f'number1 {number1:=^6.2f}') # 2 decimals, aligned center, with leading = for filler
```

Data science

```
import statistics

grades = [85, 72, 85, 96, 99, 85, 65]

statistics.mean(grades)
```

Software

- Writing code is not like letter-writing
- There should be careful planning and design
- Program development is like 'accretion' - writing code incrementally
  - Start with the simplest version of the system that runs
  - Use dummy routines - place holders in the beginning
  - Change the dummy routines to real routines - one by one
  - Add little-bit of code, run it,
    - see an error, fix the code and run-it again
    - As long as you are seeing different error messages, you are progressing